Quantum Algorithms

Daniel S. Abrams

Abstract— **This thesis describes several new quantum algorithms. These include a polynomial time algorithm that uses a quantum fast Fourier transform to find eigenvalues and eigenvectors of a Hamiltonian operator, and that can be applied in cases (commonly found in ab initio physics and chemistry problems) for which all known classical algorithms require exponential time. Fast algorithms for simulating many body Fermi systems are also provided in both first and second quantized descriptions. An efficient quantum algorithm for anti-symmetrization is given as well as a detailed discussion of a simulation of the Hubbard model.**

In addition, quantum algorithms that calculate numerical integrals and various characteristics of stochastic processes are described. Two techniques are given, both of which obtain an exponential speed increase in comparison to the fastest known classical deterministic algorithms and a quadratic speed increase in comparison to classical Monte Carlo (probabilistic) methods. I derive a simpler and slightly faster version of Grover's mean algorithm, show how to apply quantum counting to the problem, develop some variations of these algorithms, and show how both (apparently distinct) approaches can be understood from the same unified framework.

Finally, the relationship between physics and computation is explored in some more depth, and it is shown that computational complexity theory depends very sensitively on physical laws. In particular, it is shown that nonlinear quantum mechanics allows for the polynomial time solution of NP-complete and #P oracle problems. Using the Weinberg model as a simple example, the explicit construction of the necessary gates is derived from the underlying physics. Nonlinear quantum algorithms are also presented using Polchinski type nonlinearities which do not allow for superluminal communication.

Contents

Acknowledgments

I thank first and foremost my thesis advisor, Professor Seth Lloyd, who has been both a source of inspiration and a source of ideas, who taught me many important and little known concepts (related to quantum computing), who sent me to conferences across the country and around the globe and who introduced me to the leading members of the community, who was an understanding and astute co-author, who provided a constant stream of advice and insight, and (perhaps most of all) who was an all-around great guy.

I also thank my co-supervisor, Professor John D. Joannopoulos, who not only taught me about condensed matter physics, the theory of PBGs, and the art of scientific computing, but who taught me how to be a professional scientist. I thank him also for allowing me the freedom to pursue my intellectual curiosity where it led (quantum computing), for providing me with support and office space and a place to call home within the department (and a place to call home on Thanksgiving), and for many good laughs and much good advice.

I thank Professor Robert B. Laughlin (my undergraduate thesis advisor) for teaching me how to think about physics,

for sharing countless hours with me, for continuing to keep an eye on me and an eye out for me (all the way from the other coast), and for the dispensing of wisdom; in short, for being a mentor and a friend.

I thank Professor Douglas D. Osheroff for inspiration when I was an impressionable Stanford freshman and throughout my undergraduate years; without his influence, I might not have devoted myself to the study of physics.

Many members of the general quantum computing community have collectively taught me a great deal, through various conversations, some shorter and some longer. These include Leonard Adleman, Wim Van Dam, Jon Dowling, Chris Fuchs, Daniel Gottesman, Richard Jozsa, Andrew Landahl, Norm Margolus, John Preskill, Peter Shor, Mike Sipser. I am sure there are others that I have forgotten, but I am no less appreciative.

I thank all the professors and teachers I've had over the years for providing me with the necessary knowledge and for the inspiration to study. These include D. Spooner, W. Dodge, G. Munley, R. Rhodes, R. Whipple, and W. Bauer. I'm particularly grateful to Professor Paul Cohen.

I thank Colin Williams, with whom I collaborated on the work discussed in Chapter 4, and who provided on multiple occasions the opportunity to visit NASA-JPL and the California Institute of Technology.

I thank Wati Taylor, who, one day after a colloquium in building 12, introduced me to the field of quantum computation. I also thank Professor Charlie Marcus for advice in my career and otherwise.

I thank my (non-physicist) friends for putting up with me, entertaining me, supporting me, distracting me, etc. I am especially grateful to count among them Barry, Bob, Daniel, and Kelly, Jordan and Laura, Mike and Adelle and Mike, Gary and Maria, Lisa, Viviana, Brian, Judd, Alex and B., Roberto, and Charlie. I also thank those friends of mine who may not be here in Boston but who have been there for me nonetheless.

I thank all the members of the group for their friendship and advice, but particularly Nikolaj, Attila, Pierre, Shanhui, Kyeongjae, Rodrigo, Ickjin, Steven, and Tairan.

I thank also my friends on the CMT corridor, especially Dicle, Alkan, Sohrab, Claudio, Steve Simon, and Professor Tomas Arias.

I thank our department secretary Margaret O'Meara, and Peggy and Pat in the physics office, all of whom were always helpful and friendly.

I thank Ray Hagstrom for sharing with me the magic of physics when I was young.

I'd like to acknowledge several individuals with whom I have had helpful conversations directly related to the work contained in this thesis. These include I. Singer, C. Froese Fisher, W. R. Johnson, J. Jacobson, S. Simon, T. Arias, S. Johnson, I. Park, and T. Wang.

Almost last, but certainly not least, I thank my family, and especially my parents, for so much that there would be no place to begin and no space to finish.

Finally: This thesis would not have been possible without funding from several sources. These include an NDSEG graduate fellowship; grant # N00014-95-1-0975 from the Office of Naval Research; the ARO and DARPA under grant # DAAH04-96-1-0386 to QUIC, the Quantum Information and Computation initiative; a DARPA grant to NMRQC, the Nuclear Magnetic Resonance Quantum Computing initiative; the NASA/JPL Center for Integrated Space Microsystems; and, the JPL Information and Computing Technologies Research Section.

I. Introduction

A. Motivation

It is often difficult to see the hidden assumptions on which the foundations of science are built. The 20^{th} century has witnessed at least two revolutions in physics, and in both of these revolutions not only were new laws discovered, but prior assumptions were found to be false. Ideas that were previously unquestioned were revealed to be mere approximations.

The field of computer science has for the greater part of this century been based upon the ideas of classical physics. In fact, the theory of quantum computing has flourished only in the last few years. In retrospect, this may seem to be hopelessly naive — and, perhaps, to the physicist who has grown accustomed to dealing mostly with quantum mechanics and only occasionally with classical mechanics — this may appear to be somewhat surprising. The world is properly described by quantum mechanics; since computers are physical devices that exist in the world, surely they too must be properly described by quantum mechanics. The logic is obvious and compelling.

However, it is not so clear how a quantum mechanical computer would operate, or even how computation is supported by quantum mechanics[1]. Moreover, it is not at all obvious, a priori, that a quantum mechanical computer should have any advantage over a classical computer. It is perhaps for some of these reasons that the theory of quantum computing took a long time to develop. Another explanation may be that theoretical computer science is an abstract discipline, dealing with Turing machines and automata, proofs and lemmas, and mathematical models that are far removed from the actual physical reality of computing machines. Stated differently, one might say that the theory of computer science is a subject that has been traditionally much closer to mathematics than physics, and the assumption that information is classical was hidden in the abstractions.

Whatever the reason may be, one fact is now clear: the introduction of quantum physics to computer science changes fundamentally the foundations of the subject. Computers are *not* classical devices.

Often, in conferences and review articles and other such places, the subject of quantum computing is motivated by

[1] Indeed, were it not for the fact that the existance of computers in the real world implies that it must be possible to compute with quantum mechanics on some level, it would not even be obvious, a priori, if quantum mechanics should allow for the possibility of computing. Proof by existance is vastly different from an understanding of computation on a detailed, microscopic level.

the argument that transistors are becoming smaller and smaller, that pretty soon quantum effects will start to cause limitations on our ability to design faster chips, and that one should therefore study the quantum mechanics of computing. While this type of argument may have been a historical motivation, it is today so far from the point that it practically misleads. While it is surely important to investigate the quantum mechanics of small classical components, the subject of quantum computing is not merely an investigation into the possibility of smaller and faster computers. It is a new formalism for the theory of computer science. And just as in the case of mechanics, the classical theory is left as an approximation to be used in those special cases where it may be appropriate.

My own interest in quantum computing, like that of many others, came as a result of Peter Shor's now famous 1994 discovery of a quantum polynomial-time factoring algorithm[86]. This result publicized the idea that a quantum computer might be fundamentally more powerful than a classical computer. It indicated that a quantum computer would be *useful* — that it might possess capabilities that no classical computer could.[2] It also left open two important questions:

1. How might one build a quantum computer?

2. What can one do with a quantum computer other than factor large integers?

This thesis attacks the second of these questions. In particular, it asks: What can a physicist do with a quantum computer? It also explores the relationship between physics and computation in some more depth, and in particular, looks at how computational complexity theory depends very sensitively on physical laws.

The remainder of the thesis is organized as follows: The first two chapters constitute a review of computation and the physics of computation. Theoretical computer science and classical computation will be discussed in this first chapter, and quantum computing will be discussed in the second. While these two chapters provide the necessary background for the remainder of the thesis, the reader who is familiar with these topics may freely pass them over. In the final three chapters, I present new results. Chapter 3 discusses the use of quantum computers for physics simulations and *ab initio* calculations. I describe polynomial time quantum algorithms for finding eigenvalues and eigenvectors of a Hamiltonian operator, for simulating many body Fermi systems in general and the Hubbard model in particular, and for antisymmetrizing the state of the computer. Chapter 4 describes how one can obtain a quantum speed-up when calculating integrals. I derive a simpler and slightly faster version of Grover's mean algorithm, show how to apply quantum counting to the problem, develop some variations of these algorithms, and show how both (apparently distinct) approaches can be understood from the same unified framework. Finally, in Chapter 5, I'll show how the power of quantum computers depends very sensitively on the underlying physical model - and in particular, that if the physics of the universe were only slightly different than we think it is (and it might be!), then quantum computers would be unbelievably powerful.

B. An introduction to the theory of computing machines

It is necessary to understand the classical theory of computing machines before one can examine the quantum theory. Moreover, if one is to argue that quantum computers are truly more powerful than classical computers, then it will be necessary to be somewhat precise about the classical theory. This section is thus intended as a physicist's introduction to computer science (or at least, those aspects of computer science which are relevant to the work in this thesis). Since physicists have widely disparate levels of familiarity with computer science, it is written to be self-contained, with little or no prior knowledge assumed.[3]

The first significant work in theoretical computer science took place in the early part of this century. Gödel, Turing, Church, Post, Kleene and others were interested in the study of what was computable, in principle. Now it would seem that in order to address this question, one would have to specify the precise capabilities of the computer involved. However, it turns out that if you consider machines with a certain minimum computing capability, then all models of computation are equally powerful (in the sense of computability). This idea was formalized in 1936 by Church and Turing and is known as the Church-Turing thesis:

Church-Turing thesis: Any computing device can be simulated by a Turing machine.

This is an important idea, because it means that we need not study every possible type of computing machine - we need only study the model invented by Alan Turing. If something is computable (in principle), it is computable on a Turing machine. Note that the Church-Turing thesis is a thesis and not a theorem. In order to prove this conjecture, one would have had to define precisely a model for computing devices. But the whole point of the thesis is that it applies to *any* computing device, and so we don't want to limit ourselves by specifying exactly what a computing machine must be. Nevertheless, the hypothesis is fairly self-obvious and in 63 years, no one has ever conceived of a counter-example. Moreover, there are many models of computation which *have* been proven to be Turing-equivalent, including those of Church, Post, Von Neumann, and others. So it pays to examine a Turing machine more closely.

[2]For the reader who is unfamiliar with public key cryptosystems, it is important to be aware that the security of RSA [84], one of the world's most common methods for encrypting communications, depends upon the difficulty of factoring large numbers. A real life quantum computer, therefore, would render most secret codes obsolete. (Moreover, there is some indication that quantum computers would render all public key systems obsolete). This application has been one reason that quantum computing has attracted wide-spread attention.

[3]The material in this section and the remainder of this chapter, unless explicitly referenced, is part of the common knowledge. The interested reader may find good sources of further information in the following: Sipser [89], Feynman[48], Preskill[83], Shor [87], Benioff[15], Ekert and Jozsa[44], and Garey and Johnson[52].

A Turing machine consists of a head, which can be in one of a finite set of states Q, and a tape, which extends infinitely in both directions, and on which the head can read or write a set of symbols or characters from an alphabet A.[4] The head is always pointing to one particular character on the tape, and can move left or right one step at a time, to the previous or next symbol on the tape. Finally, there is a rule which tells the computer what to do at each step, based upon the state of the head and the symbol to which it is currently pointing. This rule is a function g of the form

$$g : Q \times A \Longrightarrow Q \times A \times \{L, R\} \tag{1}$$

That is, it maps a state of the head and symbol on the tape into a new state of the head, a new symbol on the tape, and an instruction to move to the left or to the right. This function therefore determines the behavior of the computer. Of course, the computer must begin with the head in an initial state and with an initial finite input string of symbols on the tape. By choosing various sets Q and A, various functions g, and the initial conditions, one can make a Turing machine compute whatever one desires (as long as it is computable in principle).

There are two important results pertaining to Turing machines that we shall consider. The first is that there exists what is known as a universal Turing machine. A universal Turing machine is one which has the capability of simulating the behavior of any other Turing machine (and therefore any other computing machine in general), simply by modifying the initial input configuration of the tape. That is, the function g (and hence the set of states Q and alphabet A) is to remain fixed. Moreover, what is perhaps more remarkable than the existence of a universal Turing machine is that it need not be very complicated: roughly 10 or 20 states and 10 or 20 symbols will suffice.[5]

The second result of interest is that there are certain problems which cannot be solved on a Turing machine, and are therefore uncomputable. The most famous example is called the halting problem[93]. Normally, there is a special state of the head Q_f which is the final, or halting state. Once the head reaches this state, the computer stops. The question is this: given a description of a Turing machine and the initial input on the tape, can you determine if the machine will ever halt? It turns out in general that you cannot. (More precisely, you cannot build a computer which can solve the halting problem). The proof is simple and runs as follows: Imagine a Turing machine H that solves the halting problem such that when given as input a description D of another Turing machine (and its input), H will halt if D does not halt, and H will go into an infinite loop if D does halt. Now what happens if you provide H as input to H? The machine can neither halt nor not halt: hence, one must conclude that it is not possible to design H in the first place. The halting problem is therefore undecidable.

One sees therefore that the Turing machine is a very useful concept: by analyzing Turing machines, one can (attempt to) determine what is in principle computable and what can never be computed. However, in real life, one is generally interested in more than whether or not something can be computed in theory: one wishes to know how long it will take, how much it will cost, etc. It may be possible, in theory, to perform an *ab initio* simulation of each and every particle in my brain — but it is hard to imagine that it will ever be possible in practice. Hence, as real life computers became more and more prevalent, the theory of computer science became more concerned with how *long* a computation would take. One thus arrives at a modified form of the Church-Turing thesis:

Church-Turing thesis (Strong Form): Any reasonable computing device can be simulated by a Turing machine in a number of steps which grows as a polynomial function of the resources used by the device.

Like the Church-Turing thesis in its weaker form, the strong form cannot be proven. (Although it has been proven for just about any specific classical model of computation that one can imagine). The statement is deliberately vague, with words such as "reasonable" and "resources" that are open to interpretation. Still, it is clear when a computer is and when a computer is not reasonable. Unreasonable machines would include those that perform an infinite number of steps in a finite time (for example, the Zeno's paradox inspired device that performs each successive operation in half the time of the previous operation), or machines that require infinite precision (for example, by storing an arbitrarily precise real number in the frequency of a single photon). Similarly, it is clear in practice what one should count as a resource. Certainly time and memory space are resources; precision or power may also be. The important point to remember, especially when one examines non-traditional computing devices (such as quantum computers), is that one must be careful to consider all the resources involved.

Essentially, the strong form of the Church-Turing thesis says that all computing devices are polynomially equivalent. This is important because it naturally divides problems into two classes: those that can be solved in polynomial time, and those that cannot. (A precise definition of polynomial time will be furnished below). If you provide an algorithm that runs in polynomial time on any one particular computer, then this algorithm can be run in polynomial time on *every* computer. Likewise, a problem that requires an exponential number of steps on any particular computer will require exponential steps on *every* computer.[6]

[4] Actually, what I've described here is only one example of a Turing machine. There are many variations of Turing machines (for example, there can be multiple tapes), but they're all equivalent, so it doesn't matter which one you choose.

[5] The exact number depends on how the machine is designed. Actually, two states will suffice if there are many symbols, and vice versa.

[6] It's worth noting that one experiences these theoretical distinctions in real life. For example, by describing an algorithm as $O(n^2)$, I have already ignored the constant speed difference (between, for example, my Pentium 90Mhz and my Pentium 400Mhz). Moreover, it may be that on the Cray C90, which can process a whole vector of numbers at a single time, this algorithm may not only run faster, it may scale more like $O(n)$, as opposed to $O(n^2)$ (at least for n within a certain

This division between polynomial and exponential problems is also convenient because it corresponds intuitively with our notion of efficient and inefficient algorithms. An algorithm that requires $O(n)$ operations is certainly faster than one that requires $O(n^3)$, but both scale quite modestly compared to an algorithm requiring $O(2^n)$ operations. In the later case, one does not need very large n before the problem becomes impossible in reality, even if it is computable in theory.[7] Polynomial-time algorithms are therefore considered tractable; exponential algorithms are considered intractable.

For both of these reasons, one is frequently concerned with distinguishing polynomial-time algorithms from those that require exponential time[8]. Hence, one must have a precise way of classifying algorithms. We say that a problem can be solved with $O(f(n))$ operations if, *in the limit of large* n, the required number of steps is bounded by $c * f(n)$, where c is a fixed constant, and n is the length of the input string. Since it is not very useful to describe an $O(n)$ algorithm as $O(2^n)$, we normally try to find the smallest $f(n)$ which the required number of steps will approach asymptotically. Note, however, that as this definition depends upon the length of the input, the computational complexity appears to depend upon the manner in which the input string is written. We must therefore add an additional constraint, that the input must be reasonably compact. For example, one can easily factor a number N in roughly $O(\sqrt{N})$ operations — but because one would normally write down N with only $n = \log N$ digits, this algorithm is actually exponential in the size of the input. One could specify the input to the problem so that it required a string of length N instead of length n (for example, by indicating the value with a string of N zeroes followed by a single one), in which case a linear time algorithm would be straightforward. However, this would not be considered reasonable.

The set of all problems whose complexity is asymptotically bounded by a polynomial function is called P. Curiously, while there are many problems known to be in P, there are hardly any decidable problems which are known not to be in P. This is because it is very difficult to prove that a problem cannot be solved in polynomial time. Thus one is forced to make such statements as "there is no known classical algorithm for factoring in polynomial time". This is also why it is difficult to prove that a quantum computer is more powerful than a classical computer. Although we know a polynomial time quantum algorithm for factoring, and even though the problem is thought not to be in P, there may in fact be a polynomial time classical algorithm that no one as yet has been clever enough to discover.

A broader class of problems, which contains many well-known problems that are thought to be exponential, is called NP. The phrase NP stands for "non-deterministic polynomial" and means (strangely enough) that such problems can be solved in *polynomial* time on a *non-deterministic* Turing machine. Not surprisingly, a non-deterministic Turing machine is not a "reasonable" model of computation, as discussed above: it is a mathematical model, with convenient properties. Essentially, it is an ordinary Turing machine which, at each step, can branch into multiple states at the same time. Pretty soon, there are an exponential number of Turing machines following an exponential (and rapidly growing) number of paths. It is said to solve the problem if only one single path finds a solution. Clearly, one would expect that an ordinary Turing machine (or any "reasonable" computer) would required an exponential number of operations to perform the same computation.

One can show that there is an equivalent (but less precise) definition of the class NP which is the following: The class NP is the set of problems for which it is possible to verify a potential solution in polynomial time. For example, the factoring problem is believed to be exponential — but once provided with potential factors, it is quite easy to multiply them and verify if they truly are factors, in polynomial time. Factoring is therefore in the class NP. In fact, almost all hard (i.e., exponential) problems that occur in practice are in the class NP. (One important exception is the class #P, to be discussed beow). Amazingly, there are no problems that are known to be in NP and not in P. We therefore arrive at the great outstanding question of theoretical computer science:

Does P=NP?

Of course, almost everyone believes that $P \neq NP$, but no one knows for sure.

In 1971, Stephen Cook showed that a certain problem, the satisfiability problem, or SAT problem, is as hard as any problem in NP[32].[9] More precisely, he showed that there is a polynomial time reduction from any problem in NP to SAT. Hence, if one could find a polynomial time algorithm for SAT, one could solve any problem in NP in polynomial time. Moreover, if there exists just one problem in NP that requires exponential time, then SAT must as well. For this reason the SAT problem is called NP-complete. Later, in 1972, Karp showed [60] that many other well known problems, such as the traveling salesman problem[10], are also NP-complete. A huge amount

range — and all real computers with finite memory always have a limited range). But no reasonable computer will require exponential time for this problem.

[7]John Preskill calculates that factoring a 400 digit number would take 10^{10} years (roughly the age of the universe) using the best known classical algorithm on state of the art computers, which can factor a 130 digit number in about one month (using hundreds of workstations). On the other hand, if we had a quantum computer that could factor a 130 digit number in one month, it could factor the 400 digit number in a few years — about 10^{10} times faster!

[8]Of course, a problem can be worse than polynomial but better than exponential, e.g., $n^{\log n}$. However, it seems that most problems are not in this intermediate range - and those that are, are usually (perhaps too casually) referred to as "exponential" even when they are not.

[9]There are various versions of the SAT problem, but the basic idea is the following: consider a boolean expression made up of n variables; for example: (b_1 or b_2) and not (b_1 and b_3 or not b_4). The question is, are there a set of values for b_1. . .b_n such that the expression evaluates to true (i.e., is "satisfied"). It turns out that, in the worst case, there is no known algorithm that improves upon merely trying all of the (exponentially many) possibilities.

[10]The traveling salesman problem is the following: consider a sales-

of work has been done on the theory of NP-completeness since Cook and Karp, and now hundreds of problems are known to be NP-complete[52]. A polynomial time solution to any one of them would imply that P=NP (and a great deal of fame). Conversely, once a problem is shown to be NP-complete, it is clear that a polynomial time solution is unlikely to be found. Interestingly, it turns out that practically every naturally occurring problem in NP (and most naturally occurring problems are in NP) is either in P or is NP-complete. The class NPI (for NP intermediate) appears to be quite lonely. In fact, this author is aware of only three problems that are thought to be in NPI: integer factoring, graph isomorphism, and certain versions of the short vector problem. This is unfortunate because problems in NPI are good candidates for quantum algorithms: they are not as hard as NP-complete problems, but they still (are thought to) require exponential time on a classical computer.

Finally, there is one last class of problems which will be discussed here, known as #P. Problems in the class #P include those that ask for the exact number of solutions to an NP complete problem: for example, how many (of the $n!$) paths that visit each of n cities have length less than l? It is self-evident that the class #P contains the class NP, and intuitively, it appears to be much larger. (But this is of course an open question). The class #P will be addressed later in the context of nonlinear quantum algorithms.

C. Classical gate arrays

Although the theory of computational complexity is machine independent, it is often useful to have a particular computational model in mind. This is especially true when considering the physics of computation. For these purposes, the classical gate array (and later, quantum gate array) is convenient.

Every computational problem can be viewed as a function that maps an input string (which we shall represent in binary) to an output string (which may also be represented in binary). In fact, one can consider each bit of the output string separately, so that a computer program can be viewed quite generally as a function f which maps n bits into a single bit:

$$f : B^n \Longrightarrow B \tag{2}$$

where B is the set of possible boolean values $\{0, 1\}$. A logic gate is just a simple computation; for example, the NOT gate is the function on a single bit such that

$$NOT(a) = \left\{ \begin{array}{ll} 0; & a = 1 \\ 1; & a = 0 \end{array} \right\} \tag{3}$$

Two bit gates are just slightly more interesting; for example

$$AND(a, b) = \left\{ \begin{array}{ll} 0; & (a = 0) \; or \; (b = 0) \\ 1; & (a = 1) \text{ and } (b = 1) \end{array} \right\} \tag{4}$$

Once we have a set of logic gates, we can begin to build more complicated computations by designing a *gate array*. In a gate array, we apply a sequence of logic gates, one after another, to the original input bits or to the outputs of the proceeding gates. For any given algorithm, the design of the gate array is fixed; we vary the input bits, the gates are applied in series, and the result is obtained. Of course, if all of our gates return only a single bit, we will not be able to build a very interesting (or very long) computation. So we will need some gates that return more than one value; for example, $COPY$ (or $FANOUT$):

$$COPY(a) = a \otimes a \tag{5}$$

An important and natural question that arises is the following: is there a universal set of gates? In other words, is there a set of gates from which one can build any computation, as long as the gates are applied correctly? In fact, the set of gates described above is a universal set: with AND, NOT, and $COPY$, one can compute any function that is computable.[11]

Since the gate array has been shown to satisfy the weak form of the Church-Turing thesis, one may ask if it satisfies the strong form. In this case, the relevant computational resource is the number of gates. How does the number of gates scale with the length of the input? One needs to be careful here, because any particular gate array is of a fixed size: if the length of the input is increased, then a larger gate array will be required. But it may be that the designer of the gate array can perform some of the work required to solve the problem. Hence we require that the gate array is sufficiently regular - that is, that the gate array can be designed with a polynomial-time algorithm. With this restriction, it is possible to prove that gate arrays and Turing machines can simulate each other with only polynomial overhead. A polynomial time algorithm requires a polynomial size gate array, and vice versa. Thus, one can prove that gate arrays satisfy the strong form of the Church-Turing thesis.

D. Physics of classical computation

As has been emphasized throughout this thesis, computers are physical devices and must obey the laws of physics. One is thus naturally driven to consider if there may be physical limitations to the power of computing machines.

man who must visit a set of cities, and would like to do so in the most efficient manner. Given a list of cities (and their locations), find the shortest path which visits each of the cities. This is often cast in an alternative manner, as a decision problem, by asking if there is a path of length d or less (that visits all the cities).

[11] It is easy to see why these gates are universal. First, recall that a computation is just a function of the form $f : B^n \Longrightarrow B$. The set of possible input strings is therefore divided into two classes: those that return true, and those that return false. Thus, the computation returns false except for on a certain (enumerable) set of possible inputs. With AND gates and NOT gates it is easy to determine if the input is any particular string. With OR gates, we can then determine if the input is a member of the set which should return true. While this technique is by no means efficient, it shows that the gates are universal.

This consideration appears to become more and more substantive as computers become smaller and faster. How long can Moore's law[12] continue? Will quantum mechanics *interfere* with our ability to compute?

If one looks again at the gates described in the previous section, an interesting and important observation can be made: the usual fundamental logic gates are irreversible. Given the output of an AND gate, for instance, it is impossible to determine what inputs were provided. Of course, the gate as described previously takes as input two bits and generates as output only a single bit, so it obviously cannot be reversible. But even if the gate is modified to output two bits - that is, the output of the AND function and an additional output bit for the purposes of reversibility - it still cannot be made reversible. In the case of AND, there are three inputs - (0,0) , (0,1), and (1,0) - which all generate the same output (0) for the AND operation. A single additional output bit is therefore insufficient to make this gate reversible. Indeed, one will find that this is the case for all of the standard logical operations.

The irreversible nature of standard logic gates leads one to an important thermodynamic consideration. Since the action of each logic gate must contract phase space (or, alternatively, decrease the system's entropy), there must be a corresponding entropy increase taking place outside the computer. This therefore leads one to the (incorrect) conclusion that each fundamental computer operation requires one to do work.

In 1961, Rolf Landauer clarified and corrected the previous statement[62]. What requires energy is not logical operations, but erasing information. One must do work equal to $k_B T \ln 2$ for each bit of information that is discarded. This is known as Landauer's principle. In the previous example, a single bit is lost during the AND operation, and so the result is the same ($k_B T \ln 2$ heat generated). However, I'll show in a moment that it is possible to perform reversible logic, in which case there is no fundamental energy limitation per gate. But it is worth noting that ordinary computers do not employ reversible logic, and that the limit of $k_B T \ln 2$ per operation could eventually pose problems. While even the tiniest and most efficient computers today are still many orders of magnitude from this limit, it should be obvious that if the size of transistors continues to shrink at an exponential rate, it will not take long before thermodynamic considerations will dominate. Eventually, computers will need to be made with reversible logic if they are to be made faster.[13]

The next critical step came in 1973 when Charlie Bennett showed that all computations can be carried out with reversible logic gates[16][17][18][19]. The number of steps taken by the reversible computer was a polynomial function of the number of irreversible operations required, and, depending upon the model of computation used (i.e., one-tape Turing machine, multi-tape Turing machine, gate array, etc.), the overhead could be only a constant factor. Because reversible classical computation is closely related to quantum computation, we'll examine reversible logic gates in more detail.

First, note that the NOT gate is reversible, and that in general, a reversible gate must have as many outputs as it has inputs. While AND and OR cannot be made into reversible 2-bit gates, the XOR gate can be cast in a reversible form:

Input	*Output*
(0,0)	(0,0)
(0,1)	(0,1)
(1,0)	(1,1)
(1,1)	(1,0)

(6)

Feynman calls this XOR gate a "controlled-NOT", because it flips the second bit if and only if the first bit is 1. The controlled-NOT is a convenient gate, and one employed frequently in quantum computations. Unfortunately, it is not universal, i.e., the controlled-NOT is insufficient for building an arbitrary classical computation. In fact, one can prove that no set of (classical) two bit reversible gates is universal. Fortunately, one needs only a single 3-bit gate in order to perform universal reversible logic. Two famous examples of 3-bit gates that suffice are the Fredkin and Toffoli gates [51]. The Fredkin gate has an additional curious property in that the number of ones and zeros does not change between the input and the output. This was useful for Fredkin, who was interested in showing how one could build a universal computer with a classical "billiard ball" model.

The Fredkin gate can be thought of as a controlled exchange: if the first bit is 1, then exchange the positions of the second and third bits. Otherwise, do nothing. The truth table appears as follows:

Input	*Output*
(0,0,0)	(0,0,0)
(0,0,1)	(0,0,1)
(0,1,0)	(0,1,0)
(0,1,1)	(0,1,1)
(1,0,0)	(1,0,0)
(1,0,1)	(1,1,0)
(1,1,0)	(1,0,1)
(1,1,1)	(1,1,1)

(7)

If one is given a supply of fresh ones and zeroes, the Fredkin gate can be made to simulate any of the usual irreversible two bit gates. With this in mind, it is relatively easy to see how one can replace ordinary irreversible logic gates with reversible gates such as the Fredkin gate. In each step of the computation, extra "garbage" bits are introduced. These bits are not required for future steps, but only so that one can retrace the backward path, to satisfy the constraint of reversibility. However, it appears that the number of garbage bits will grow quite quickly (possibly, one for each step of the computation), and this

[12]Named after Gordon Moore of Intel, it states that computers become twice as fast roughly every 18 months.

[13]If the current pace of progress continues, the fundamental limit of irreversible computation will be reached in roughly 60-80 years.

led some early critics to suggest that reversible computers were in some sense hiding entropy in the garbage bits. Fortunately, there is a useful trick (due to Bennett) which prevents garbage build-up[19]. For any computable function f acting on a bit string $\vec{x}$, one can always perform the following transformation reversibly:

$$(\vec{x}, \vec{0}) \Longrightarrow (\vec{x}, f(\vec{x})) \tag{8}$$

The transformation is obviously reversible on a macroscopic scale: since the bit string $\vec{x}$ is part of the output, one can easily reconstruct the input. The problem is, can one transform an ordinary irreversible algorithm for calculating $f(\vec{x})$ into a reversible one such that it does not generate any extra garbage bits (as in the equation above). Bennett's trick is the following: follow the course of the original irreversible algorithm, generating garbage bits along the way, to maintain reversibility. Once the answer has been obtained, make a copy of the answer. (This is easily accomplished with a sequence of controlled-NOT gates). Then reverse all the steps which were used in the original calculation, thereby cleaning up the garbage. One is thus left with simply the original input $\vec{x}$ and the copy of the answer $f(\vec{x})$, as desired. This simple trick is quite useful, and we shall employ it frequently (and variations thereof) in the design of quantum algorithms.

Because of the work of Bennett, Fredkin, and others, we have a detailed understanding of how to perform arbitrary computation using the fundamental principles of classical mechanics. It was therefore natural to investigate how computation is possible with quantum systems. Paul Benioff, in a series of important papers, first showed how to design quantum mechanical models of computation[11][12][13][14]. Feynman [47][46] also did important work in this area, as did Margolus [72][73][74]. They described how one could build Turing machines, gate arrays, and cellular automata, and showed that, with the proper Hamiltonian, the systems could be made to time-evolve in such a way that a computation was performed. Feynman [46] was the first to speculate that a quantum computer might in principle be more powerful than a classical computer: in particular, he thought that perhaps a quantum computer could be used to efficiently simulate other quantum systems (a conjecture later proven by Lloyd [68]). Thus one arrives naturally at the topic of quantum computing, which will be discussed in the following chapter.

II. Quantum Computers

A. Quantum bits

[14] The fundamental unit of quantum information is the quantum bit, or qubit[85][15]. Mathematically, a qubit is simply a ray in a two-dimensional Hilbert space: it represents the quantum state of a two state system, just as a classical bit represents the classical state of a two state system. Physically, a qubit may be stored in many different ways: for example, with a photon polarization, two states of an atom, or the spin of a particle. Whatever the actual physical system may be, we shall always write the two states as $|0\rangle$ and $|1\rangle$, thus indicating the correspondence between classical and quantum bits.

The general state of a single qubit is $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$; since $|\alpha|^2 + |\beta|^2 = 1$, and the absolute phase is not physically observable, the general state is described by two real numbers. A classical bit, on the other hand, is described by (simply) a classical bit. Even an analog or "fuzzy" bit requires only a single real number for its description. Still, it is not apparent how much of the information in the qubit is truly "available", and even with a single quantum bit one can see that the question of how one "gets the information out" is not trivial. Given many copies of the state $|\psi\rangle$, one can make a series of measurements to determine (or approximate) α and β. However, with a single preparation, information will be lost during the measurement process. One might be tempted to think that by being sufficiently clever one could take a single preparation of $|\psi\rangle$ and make multiple copies, but this is not possible. There is a theorem due to Wooters and Zurek [100] which says that one cannot copy a quantum state (the "no-cloning" theorem).

The situation becomes more interesting when one has multiple qubits. While a single two-state system is described by a wavefunction of the form $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, a pair of two-state systems is described by a ray in the tensor product space:

$$|\Psi\rangle = \alpha_{00}\,|0\rangle \otimes |0\rangle + \alpha_{01}\,|0\rangle \otimes |1\rangle + \alpha_{10}\,|1\rangle \otimes |0\rangle + \alpha_{11}\,|1\rangle \otimes |1\rangle \tag{9}$$

Whereas two classical bits still require only two classical bits for their description (or two real numbers if they are analog), the two qubit system is described by 6 real numbers. We see that in general, an n qubit system is described by $2^{n+1} - 2$ real numbers. For those who are interested in calculating the properties of quantum systems, this exponential growth is unfortunate; but for those who are interested in using quantum systems to calculate, this property provides the first step toward a more powerful computer.

The 2^n different basis vectors which are generated by taking tensor products of the states $|1\rangle$ and $|0\rangle$ form the canonical basis. There are a few different notations to describe these basis vectors. Using one of the 16 basis states for $n = 4$ as an example, we may write, depending on the situation,

$$\begin{aligned} |\Phi\rangle &= |0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle && (10)\\ &= |0\rangle\,|1\rangle\,|1\rangle\,|0\rangle && (11)\\ &= |0110\rangle && (12)\\ &= |6\rangle && (13) \end{aligned}$$

where in the last equation the state is labeled according to the integer which represents the qubits viewed as a single

[14] As with the previous chapter, the material covered herein, unless explicitly referenced, is part of the common knowledge. The interested reader may find good sources of further information in the following: Sipser[89], Feynman[48], Preskill[83], Shor[87], Benioff[15], Ekert and Jozsa[44], and Garey and Johnson[52].

[15] Like the word "bit", the word "qubit" is used to refer both to an amount of information and to the physical system that registers that information.

binary number (i.e., $6_{10} = 0110_2$). This last notation is the most dense; an arbitrary state of an n qubit system is merely

$$|\Psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i |i\rangle \tag{14}$$

which is an expression that will occur frequently.

As with the single qubit system, there is a question of how one can access the information hidden in the state, only now the problem is more severe. Measuring the qubits will only reveal a single n bit integer i, with any particular result occurring with probability $|\alpha_i|^2$. There may be an exponentially large amount of information stored in the state of a quantum system, but it is not at all obvious how it is to be exploited.

B. Quantum gate arrays

Having thus described how quantum information is stored, the next issue one must address is how it is to be processed. The most natural way to compute with a set of quantum bits is with a quantum gate array.

A quantum gate is a unitary transformation acting on one or more qubits. Essentially, we imagine that the Hamiltonian for all the other qubits (upon which the gate does not act) is zero; the qubits involved in the gate will time-evolve according to their local Hamiltonian. One important but limited set of gates are those which simply map one member of the canonical basis into another; for example:

$$|00\rangle \implies |00\rangle \tag{15}$$
$$|01\rangle \implies |01\rangle \tag{16}$$
$$|10\rangle \implies |11\rangle \tag{17}$$
$$|11\rangle \implies |10\rangle \tag{18}$$

which is simply a quantum version of the controlled-NOT gate. If these basis states are ordered in the natural way (according to the binary number represented), one can write the transformation as a matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{19}$$

which is manifestly unitary. It should be easy to see that all classical reversible logic gates correspond to permutations as above; assuming that it is possible to generate any desired unitary transformation, one therefore concludes that a quantum gate array can perform any computation that a classical gate array can. The qubits are initialized in the same manner as the classical bits; the calculation is performed using the quantum counter-parts of the classical gates. Throughout the entire calculation, the state of the quantum computer is always one of the canonical basis states: it is never in a superposition of multiple basis states or an entangled state. There is no difficulty in reading the result. Thus a quantum computer can be easily made to mimic a classical computer, and the quantum computational complexity is no worse than the classical computational complexity. It follows that classical computation is (in a very real sense) just a limiting case of quantum computation.

Frequently during a quantum calculation, one desires to perform a classical computation. For example, one may wish to do a transformation such that

$$|\vec{x}\rangle\,|\vec{0}\rangle \implies |\vec{x}\rangle\,|f(\vec{x})\rangle \tag{20}$$

As long as an algorithm exists for computing $f(\vec{x})$, this transformation can be performed using quantum logic gates. Moreover, the quantum algorithm is the same as the reversible classical algorithm, which, as discussed in the previous chapter, is essentially the same as the original (presumably irreversible) algorithm.

C. Universality

Of course, permutation matrices of the sort described above are only a small subset of all possible unitary transformations. For example, the single qubit transformation

$$|0\rangle \implies \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{21}$$
$$|1\rangle \implies \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) \tag{22}$$

has no classical analogue. This transformation (and, in particular, its multi-qubit extension) is known as the Walsh-Hadamard transform. It is common for quantum algorithms to make frequent use of this gate.

A natural question to ask is whether or not there exists a set of universal quantum gates, which can be combined together to form any possible unitary transformation. Since there are an infinite number (continuum) of unitary transformations on even a single qubit, this problem is a bit more subtle than the corresponding classical problem. However, since any real physical system will posses only finite accuracy, it should be sufficient to find a set of gates which can be applied so that they will generate an n bit unitary transformation that is arbitrarily close to any desired n bit unitary transformation. Fortunately, such gates exist [6] [39][43][69]. In fact, Lloyd has shown that almost any 2-qubit gate is sufficient for generating arbitrary unitary transformations[69][16]. This later result is important because it means that one doesn't need to worry much about what type of interaction Hamiltonian exists between pairs of qubits: almost any interaction will do.

A general unitary transformation on n qubits is described by $O(2^{2n})$ complex numbers. It is therefore not too surprising that a general n qubit unitary transformation will require an exponential number of gates. Since a quantum computation is nothing more than a series of unitary transformations - which when multiplied together are just one very complicated unitary transformation - this implies

[16] It is interesting to note that only two-qubit gates are required for universal quantum computation, whereas three-bit gates were required for classical reversable logic

that most computations will require an exponentially large number of gates. One of the main goals in designing a quantum algorithm, therefore, is to see if it is possible to effect a desired n qubit unitary transformation using only a polynomial number of quantum gates.

The class of problems which can be solved by a quantum gate array of polynomial size is called BQP[17]. In order for this class to be well-defined, it is important that it should not depend upon the choice of gates; moreover, it should also correspond to polynomial-time complexity for other models of quantum computation, such as quantum Turing machines. It is clear that quantum gate arrays must be polynomially equivalent because once one has a set of (constant sized) circuits to effect any desired two qubit unitary transformation, all gate arrays will look the same. In 1993, Yao showed that quantum Turing machines and quantum gate arrays were polynomially equivalent [101]. Moreover, we know that any physically realistic quantum computer can simulate any other with only a polynomial overhead, because a quantum computer is nothing more than a physical system, and Lloyd has shown [68] that there exist universal quantum simulators. The class BQP is therefore quite robust: indeed, one could argue that the quantum version of the Church-Turing thesis is a consequence of the laws of physics as they are known today.

The formalism described previously is based upon a system composed of a set of individual qubits. However, one may wish to consider a set of three level systems (a qu-trit), or even n level systems.[18] The mathematics are likely to be more complex, but one would hope that the fundamental results should not be effected any more than in the classical case, where using trits or analog components are of no impact in terms of computational complexity. In fact, our quantum formalism is not sensitive to the makeup of the individual subsystems. The state space of the combined system is the tensor product of the state spaces for each individual component system. This is physically reasonable: one can only store so much information in a single, local, quantum system, and a reasonable model of computation must combine many such subsystems. Moreover, this is how the quantum computer gets its power — by virtue of the exponential growth in the size of the Hilbert space with respect to the number of subsystems which are combined. As long as it is possible to perform an arbitrary unitary transformation on two subsystems at a time, it is evident that the model will be universal; conversely, one can represent the state of any subsystem with a collection of one or more qubits, and since the qubits can be made to evolve according to an arbitrary unitary transformation, the more complex system will have no additional computational power. In some sense, it is the process of decomposing a large quantum computer into component subsytems that allows us to employ the gate array model.

All quantum algorithms consist of three steps: First, initialize the computer into the state $|00...0\rangle$; second, perform a unitary transformation (which is generated through a sequence of unitary transformations); third, make a measurement of the computer's state with respect to the canonical basis. From this description, one can see that is not possible for a quantum computer to perform any computations that are classically uncomputable; a Turing machine can easily simulate a quantum computer, as long as it is allowed exponential overhead [19]. The quantum theory of computation does not therefore have any impact on *what* can be computed, only on *how long* a computation will take.

D. Introduction to quantum algorithms

In 1985, David Deutsch suggested the idea of quantum parallelism [40][41]: by placing a computer in a superposition of input states, it could in some sense perform a large number of computations at the same time. Although his original work was with quantum Turing machines, the essential idea can be easily seen with the gate array models discussed thus far. Consider a quantum computer with $l = 2n$ qubits, which we will conceptually divide into two distinct registers of n qubits each. The initial state of the computer will be the zero state, that is $|\psi\rangle = |0\rangle \otimes |0\rangle \otimes ... \otimes |0\rangle$; we shall also write this state as $|\psi\rangle = |0,0\rangle$, that is, by labeling the state with the two integers that represent the two n qubit registers.

We operate on each of the first n qubits with a Walsh-Hadamard gate, thus obtaining:

$$|\psi\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes ... \otimes \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \otimes ... \otimes |0\rangle$$

$$= \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, 0\rangle \tag{23}$$

As in equation (20), one can perform a transformation $|\vec{x}\rangle|0\rangle \Longrightarrow |\vec{x}\rangle|f(\vec{x})\rangle$ for any computable function f. However, since the input register is in a superposition state, one now obtains

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, f(i)\rangle \tag{24}$$

One thus calculates the value of $f(i)$ for all 2^n possible inputs, without performing any more elementary gate operations than were required to classically calculate $f(i)$ for a single value. Unfortunately, it is not clear if this accomplishes anything: making a measurement on the system will return only one $f(i)$ chosen at random. Thus a quantum algorithm will accomplish nothing unless it exploits interference; one must perform another unitary operator after equation (24) such that the various paths interfere

[17] BQP stands for bounded error quantum polynomial time.

[18] The situation is somewhat more complicated if the quantum variables are allowed to be continuous. See Lloyd and Braunstein [70].

[19] A priori, one might assume that a classical computer would require exponential memory resources to simulate the behaviour of a quantum computer. It is thus interesting to note that a classical computer can simulate a quantum computer without requiring exponential memory, as long as it is allowed exponential time [21]. (See also [83]).

with one another. Hopefully, after this additional, inherently non-classical operation, one may be able to make a measurement on the resulting state and learn some joint property of all the $f(i)$. It took seven years to make further progress; but in 1992 Deutsch and Jozsa showed the first example of a problem in which quantum parallelism could be exploited to obtain a super-classical speedup[42]. Soon after, Bernstein and Vazirani [21] showed the first exponential separation between a quantum and classical algorithm (i.e., a problem for which there is a polynomial time quantum algorithm but no polynomial time classical algorithm).[20] A year later, in 1994, Dan Simon provided a more elegant problem with an exponential separation [88], which led directly to Peter Shor's 1994 discovery of the factoring algorithm[86].

Both Simon's algorithm and the Bernstein and Vazirani algorithm rely upon a black box function, which is called an oracle.[21] The oracle is provided an input value x (either classical or quantum) from which it computes $f(x)$; however, one is not allowed to look inside the oracle (thus "black box") to see how it does the actual computation. One wishes to determine how many calls to the oracle are required to determine a specific property of the function $f(x)$. It is possible in this context to prove that a quantum computer is more efficient than a classical computer, as in the previously mentioned papers. Such a result is said to be "relative to an oracle", and such a proof is said to be "relativised". While these relativised results are interesting, and indicate that a quantum computer probably is more powerful than a classical computer, they are not as compelling as a non-relativised speedup. (Unfortunately, while the Shor algorithm does not rely upon an oracle, it is also not proven to be faster than a classical algorithm — it is still possible that someone will discover a classical algorithm that is just as fast).

One can gain insight into the meaning of the oracle results by considering Grover's algorithm [55][56].(Grover's algorithm will be discussed in detail in Chapter 4). In the most basic version of Grover's algorithm, one is given (inside a black box) a boolean function $f(x)$ over an n bit input such that the function return zero for all values except for a single value x_0; one wishes to determine what the value of x_0 is. For this reason, it is called a "database search." With no additional information, it is readily apparent that a classical algorithm can do no better than to try values of x until it finds the solution; on average it will take time $\frac{N}{2}$, where $N = 2^n$ is the total number of possible values of x. In 1996, Lov Grover showed how this can be solved with a quantum algorithm in time $\sqrt{N}$: not an exponential speedup, but still impressive considering the fundamental nature of the problem. The problem is that the classical estimate of $\frac{N}{2}$ is predicated on the assumption that one cannot look inside the black-box. However, if the computer is to execute the code required to calculate $f(x)$, then that code must be available, so restricting an algorithm from examining the function is an artificial constraint. Once this constraint is removed, it is no longer possible to know how many operations are required to find the solution: it might be that a sufficiently clever algorithm could look at the code which calculates $f(x)$ and determine the solution right away, without having to try any values of x. Thus it is only possible (so far) to prove that a quantum algorithm is faster than a classical algorithm in a context in which both of the algorithms are in some sense restricted. On the other hand, the power of the quantum computer is readily apparent in the fact that Grover's algorithm does not need to look at the function; the $\sqrt{N}$ speedup is completely independent of the form that $f(x)$ takes. It seems clear that the quantum computer is somehow more powerful than the classical machine, even if we are unable as yet to prove it. This viewpoint is most convincingly argued by considering that — while it may be in theory possible for an efficient classical algorithm to analyze the function $f(x)$ and find the solution — it seems extremely unlikely that such an algorithm would exist for an arbitrary $f(x)$.

It is worth noting that Grover's algorithm is known to be optimal [20]. It thus follows that if quantum computers are to solve NP-complete problems in polynomial time, it will require a deep insight into the structure of NP-complete problems that can be exploited by a quantum computer. Such an insight is unlikely to be forthcoming. Indeed, just as the oracle model allows one to demonstrate super-classical speed-ups, it also allows one to demonstrate quantum lower bounds. In addition to the result just mentioned, it has been proven that the problems of mean and median estimation admit only a quadratic speed-up [76], that finding the parity of n boolean values admits only a constant speed-up [45], and that, in general, almost all functions admit only a factor of two speed increase [8]. While these results demonstrate that one cannot in general make an algorithm faster with "quantum magic", and while they do limit the possible applications of quantum computers, they also help to clarify which interesting problems have complexities that may be significantly improved.

E. Simon and Shor algorithms

Simon's algorithm [88] is a prototypical example of a quantum algorithm. Because it is not necessary for the remainder of the thesis, the reader may skip this section, if desired. It has been included, however, because it is the cleanest and clearest demonstration of the power of quantum computing. It also forms the basis of Shor's factoring algorithm [86], which will be discussed briefly at the conclusion of this section.

[20] The Deutch-Josza problem, while difficult to solve on an ordinary Turing machine, is easy to solve on a probalistic Turing machine. A probabilistic Turing machine, loosely defined, is an ordinary Turing machine that has a built in generator of random bits, and that is not required to solve the problem 100% of the time. The class of problems that can be solved in polynomial time on a classical probabilistic Turing machine is known as BPP, and corresponds most closely with our intutive notion of which problems are tractable in reality. It is interesting to note that there are a lot of open questions regarding the class BPP and its relation to other classical complexity classes; for example, does BPP = PSPACE?

[21] A black box function is not precisely the same thing as an oracle, but the two terms are used to mean the same thing in the quantum computing literature, because the black box is essentially the same as a Turing machine with an oracle.

In Simon's problem, one is given a $2 \to 1$ function that maps an n bit string into an $n-1$ bit string, with the following special property:

$$f(i) = f(j) \Longleftrightarrow i \oplus j = c \tag{25}$$

where the $\oplus$ indicates addition over the group B^{n-1}. (In other words, each bit is added independently, without carry, with $1 \oplus 1 = 0$.) (Alternatively, this addition can be seen to be a bitwise XOR of the two $n-1$ bit strings). Note that this special type of addition has the interesting property that

$$i \oplus j \oplus j = i \tag{26}$$

for any i and j. The function f is therefore periodic, in the sense that $f(i) = f(i \oplus c) = f(i \oplus c \oplus c)$, etc. One is given an oracle (black box) which computes f for any input i. The problem is to determine the value of c, with as few operations (or calls to the oracle) as possible.

It is easy to see that classically, there is no better algorithm than randomly guessing values of i and querying the oracle until the function returns the same result twice (at which point one determines c). Since there are 2^n values of i, it is also easy to see that this algorithm requires exponential time. The quantum algorithm, however, will run in polynomial time. First, we generate a state as in (24). Note that this requires only a single call to the quantum oracle. Next, we make a measurement of the second register.[22] This will result in a state of the form

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(|a, f(a)\rangle + |a \oplus c, f(a)\rangle \right) \tag{27}$$

for a particular value of $f(a)$ chosen at random. This is because there exist two values of i which correspond to every possible value of $f(i)$. Since the system is now in a superposition of two states which differ by c, it would seem to be an easy matter to determine the answer. However, measuring the state would reveal just one value, either a or $a \oplus c$, and if we were to repeat the experiment, we would obtain a different (random) value of a. The information is indeed contained within the state of the quantum computer, but it is hidden in a way which makes it difficult to extract. It is therefore necessary to do something more clever. The trick is to generate interference between the states by performing an operation such as the original Walsh-Hadamard transform on the first n qubits. First, we shall rewrite the state of the system in a slightly different form

$$|\psi\rangle = \frac{1}{\sqrt{2}} \left(|a, f(a)\rangle + |a \oplus c, f(a)\rangle \right) \tag{28}$$

$$= \left[\frac{1}{\sqrt{2}} \left(|a\rangle + |a \oplus c\rangle \right) \right] \otimes |f(a)\rangle \tag{29}$$

[22] It is not actually necessary to perform this measurement; however, it makes the algorithm appear simpler.

We factor out the pure state $|f(a)\rangle$ of the final $n-1$ qubits because it is no longer relevant to the computation. It will be omitted in what follows (i.e., we will only consider the state of the first n qubits). We now perform an operation similar to the Walsh-Hadamard transform that causes $|0\rangle \Longrightarrow \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle \right)$ and $|1\rangle \Longrightarrow \frac{1}{\sqrt{2}} \left(|0\rangle - |1\rangle \right)$.

$$|\psi\rangle \longrightarrow \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \left[(-1)^{a \cdot i} + (-1)^{(a \oplus c) \cdot i} \right] \tag{30}$$

$$= \frac{1}{\sqrt{2^n}} \sum_{i \,\in\, c \cdot i = 0} (-1)^{a \cdot i} |i\rangle \tag{31}$$

This step requires perhaps a small amount of explanation. Any state $|a\rangle$ will be transformed into an equally weighted superposition over all $|i\rangle$. However, the phases of the components are determined by $|a\rangle$: for every qubit where $|a\rangle$ is originally $|1\rangle$ and $|i\rangle$ is also $|1\rangle$, we pick up a phase of (-1). Hence the state $|a\rangle$ maps into $\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \, (-1)^{a \cdot i}$ and the state $|\psi\rangle$ transforms as above. The second step simply reflects the realization that the only terms in the sum which will not cancel are those for which $c \cdot i = 0$.

A measurement of the system will yield a state $|i\rangle$ chosen at random such that $c \cdot i = 0$. In contrast to the situation in equation (27), a measurement now reveals information about the value of c, which is the objective. Indeed, only $O(n)$ trials are necessary to determine n linearly independent vectors i such that $c \cdot i = 0$. From these, one can easily determine c. The quantum algorithm is therefore exponentially faster than the best classical algorithm.

The Shor algorithm works in essentially the same way. It has been known since 1976 that the problem of factoring n can be reduced to the problem of finding the order of $x \bmod n$, that is, the least integer r such that $x^r = 1 (\bmod n)$. If one considers the function $f(a) = x^a \bmod n$, the order r is simply the period of this function. The Shor algorithm proceeds exactly as the Simon algorithm described above, with two differences: First, the function $f(a) = x^a \bmod n$ is different from the function used in Simon's problem. Second, the Walsh-Hadamard transform used in the last step to find the period of the function over the group B^{n-1} is replaced with a quantum Fourier transform [33] that finds the conventional period over the ordinary multiplicative group (which is the order r). More detailed discussions of the Shor algorithm can be found in [86], in [87], and in the review by Ekert [44].

F. Accuracy and errors

Unlike modern digital computing machines, quantum computers are analog devices. While on a superficial level it may appear that the qubits themselves are digital, the information is stored in the phases of the states, and these are complex numbers. It is important therefore to consider what level of precision is required. For example, it is known that a classical analog device with exponential precision can solve NP-complete problems in polynomial time; however, exponential precision is not considered to

be reasonable. It is therefore important to determine what level of precision is required of a quantum computation.

A quantum algorithm is nothing more than a series of unitary operators applied to an initial state,

$$|\psi_n\rangle = U_n ... U_3 U_2 U_1 |\psi_0\rangle \tag{32}$$

followed by a measurement of the final state (made with respect to the canonical basis). Let us suppose that at each step of the computation, the unitary operators which we perform are not exactly the operators which we intend.[23] Alternatively, we can imagine that there is a little gnome who performs a small unitary transformation in between each gate. Hence, the calculation which we actually perform is

$$\left|\widetilde{\psi_n}\right\rangle = \widetilde{U_n} ... \widetilde{U_3}\widetilde{U_2}\widetilde{U_1} |\psi_0\rangle \tag{33}$$

where $\widetilde{U}$ indicates an operator which is close to U; specifically, if $|\psi_1\rangle = U_1 |\psi_0\rangle$, then $|\psi_1\rangle = \widetilde{U_1} |\psi_0\rangle + \epsilon_1 |E_1\rangle$, where ϵ_1 is small and all the states are normalized. $\left|\widetilde{\psi_n}\right\rangle$ indicates the actual state vector obtained, which will hopefully be close to $|\psi_n\rangle$, the ideal result. Applying all of the $\widetilde{U}$ in series, and collecting terms, we find

$$\left|\widetilde{\psi_n}\right\rangle - |\psi_n\rangle = \epsilon_n |E_n\rangle + \widetilde{U_n}(\epsilon_{n-1}) |E_{n-1}\rangle + \widetilde{U_n}\widetilde{U_{n-1}}(\epsilon_{n-2}) |E_{n-2}\rangle \ . \ . \ . \ \widetilde{U_n} ... \widetilde{U_2}(\epsilon_1) |E_1\rangle \tag{34}$$

The worst possible situation would occur if all the terms on the right hand side were to add in phase. This would yield

$$\left|\widetilde{\psi_n}\right\rangle - |\psi_n\rangle = (\epsilon_n + \epsilon_{n-1} + ... + \epsilon_1) |E\rangle \tag{35}$$

where $|E\rangle = |E_n\rangle = \widetilde{U_n} |E_{n-1}\rangle = ... = \widetilde{U_n} ... \widetilde{U_2}(\epsilon_1) |E_1\rangle$. Let us assume that the maximum error occurring with each gate is ϵ; then

$$\left\| \left|\widetilde{\psi_n}\right\rangle - |\psi_n\rangle \right\| \leq n\epsilon \tag{36}$$

One concludes therefore that the errors grow linearly with respect to the number of gates applied. Even without error correction, the required precision of a quantum computer would grow only linearly with the number of gates. (Note that the number of bits of precision therefore grows only logarithmically). This level of precision is acceptable in a "reasonable" model of computation.

Having thus addressed the issue of precision (and unitary gate errors), we now address errors in general. There are two types of errors which can occur during a quantum computation: bit flips (i.e. $|0\rangle \longrightarrow |1\rangle$ and $|1\rangle \longrightarrow |0\rangle$) and phase flips ($|0\rangle \longrightarrow |0\rangle$ but $|1\rangle \longrightarrow -|1\rangle$). Smaller errors (e.g., $|0\rangle \longrightarrow \sqrt{.99}|0\rangle - \sqrt{.01}|1\rangle$) are merely superpositions of the error-free state and a small amount of one (or both) of these error states. On the surface, it would seem to be impossible to correct for such errors during a quantum computation. Classical error correcting codes are based upon the idea of redundant information and frequent measurement (e.g., store the bit 1 as three bits 111; as long as only a single bit error is committed, one can correct for it). But one is unable to make duplicate copies of quantum information (the "no-cloning" theorem), and moreover, one is unable to detect errors by looking at the state of the computer during the course of the computation, because doing so would necessarily cause a "collapse" of the wavefunction. Nevertheless, Peter Shor demonstrated [29][28] that it is possible to perform quantum error correction, and a large literature has developed on this topic (e.g., [91][82]).

I will not discuss quantum error correcting codes in detail here; however, it is important to know of their existence and to understand the fundamental principles upon which they are based. The key concept on which all quantum codes depend is that errors are assumed to be uncorrelated and local. Uncorrelated implies that the errors are in some sense random: it is always possible, even with classical error correction, for a little gnome who knows your scheme to create errors in your calculation by committing just the right errors in the right places. Local implies that the errors act on only one qubit (or a few qubits) at a time. This is physically reasonable. It is likely that, during the course of a calculation, random atoms may spontaneously emit photons. It is not likely that all the atoms in your system will spontaneously emit photons at the same time. Given these assumptions, and as long as the frequency of errors is below a certain threshold, it is possible to perform arbitrarily long fault-tolerant quantum computations [82].

This concludes our discussion of the mathematical formalism used in quantum computation. In addition to providing the necessary background for the rest of the thesis, the author hopes to have made a convincing case that quantum computation is in fact a physically reasonable paradigm that poses a very real challenge to the strong form of the Church-Turing thesis. Indeed, it appears most likely that the strong form of the Church-Turing thesis is wrong.

G. Potential implementations

Although this document focuses on quantum algorithms, it would not be complete without at least some discussion of the possible hardware with which a quantum computer might be implemented. There is at this time a wide variety of suggestions for quantum computing devices; below, I will address a few of the more promising proposals.

Trapped Ions. First suggested by Cirac and Zoller[31], this scheme uses ions in a linear Paul trap to store quantum bits. The ground state of an ion is labeled $|0\rangle$ and the $|1\rangle$ state is a metastable excited state. With a properly timed laser pulse that is tuned to the energy difference, one can effect any desired single qubit unitary transformation. One

[23] The particular argument given here is similar to that in Preskill [83] and in Kitaev [61].

can measure the state of a qubit with a laser that drives the $|0\rangle$ state to a short-lived excited state $|f\rangle$, and looking to see if the ion fluoresces. Multiple ions in the trap are physically separated because of their coulomb interaction and can be addressed individually. The difficult part of this scheme (like most proposed quantum computer implementations) is with the two qubit gates. In this case, the trick is to excite a normal vibrational mode of all the ions in the trap when a particular ion absorbs a photon. Another laser pulse can then be used to generate a transformation on a physically distant qubit in such a way that it depends on the state of the first qubit.

Trapped Atoms. This scheme was proposed by Pellizzari et. al. [80]. Instead of using trapped ions, it uses neutral atoms trapped within an extremely high Q optical cavity. The single qubit operations are similar to those used in the ion trap; however, the two qubit gates are implemented somewhat differently. In this case, the atoms interact with the normal modes of the electromagnetic field inside the cavity. The electromagnetic field is then used as the mediator of information between qubits, as opposed to the vibrational modes employed in the ion trap.

Photons. An obvious place to store quantum information is in the polarization of single photons. Unlike trapped ions or atoms, it is relatively easy to keep a photon state from decohering. The problem is that it is correspondingly difficult to get photons to interact with each other. One possible technique is to use small cavity QED, as above, but use atoms to cause the photons to interact with each other, rather than the other way around. Kimble has pursued this approach [92] and has been successful in demonstrating that it is at least possible to generate substantial photon-photon couplings in this manner.

Nuclear Spins. A lot of recent attention has been generated by proposals [34][53] in which quantum computation is performed using NMR. (The essential idea is the same as that suggested by Lloyd in 1993, involving arrays of weakly coupled quantum systems [67].) RF pulses are used to control interactions between pairs of nuclear spins; because these spins are well isolated form their environments, they can have very long coherence times (on the order of seconds). The current NMR quantum computing implementations differ from the other schemes because they involve an ensemble of quantum computers which all perform the exact same computation in parallel; moreover, since it is not possible to initialize the computer in a pure state, various techniques are used to create "pseudo-pure states", in which the slight deviations from an equal distribution are exploited so that a computation may be performed. At room temperature, the unfortunate result is an exponential loss in signal strength as the number of qubits is increased; however, if the system is cooled so that $k_B T \sim \Delta E$, then one enters a new regime which does not have this problem. But at such low temperatures, one can no longer perform liquid NMR, and without the tumbling of the molecules, there are additional difficulties. It thus appears that with current technologies it will be difficult to perform computations with more than 10 or 12 qubits[96][35], although solid NMR may have the potential for large scale quantum computing. The current state-of-the-art with liquid NMR is roughly 5 or 6 qubits. However, it is not clear at this time if these experiments are truly performing quantum computations or if they are acting in a purely classical regime[26].

Semi-conductor devices. If possible, semi-conductor devices would obviously be a highly desirable platform for quantum computation. Unfortunately, it is hard to maintain coherence in solid state devices: charge dephasing times are typically on the order of nanoseconds. DiVincenzo and others have proposed using electron spins as quantum bits because the spin-coherence times may be on the order of microseconds, which may be sufficient [66][27]. Quantum logic operations are performed by adjusting the exchange coupling between spins in single-electron quantum dots, which is accomplished by varying electric or magnetic fields, or by adjusting the tunneling barrier. In a different scheme, recently proposed by Kane[59], silicon is combined with NMR in an attempt to gain the best of both worlds. Quantum information is stored on the nuclear spins of impurity ions in doped silicon, and quantum logic operations are performed by using gate voltages to control electrons which have hyperfine interactions with the nuclear spins.

In conclusion, it is at this stage quite unclear which (if any) of these proposals will lead to a useful quantum computer. What is clear — unfortunately — is that a useful quantum computer will not be easy to build, and is unlikely to exist in the near future.

III. Quantum Quantum Simulators

[24]**Summary.** This chapter discusses the possibility of using a quantum computer as a quantum simulator. It describes a new polynomial time algorithm that uses a quantum fast Fourier transform to find eigenvalues and eigenvectors of a Hamiltonian operator, and that can be applied in cases (commonly found in *ab initio* physics and chemistry problems) for which all known classical algorithms require exponential time. Fast algorithms for simulating many body Fermi systems are also provided: both first and second quantized descriptions are considered, and the relative computational complexities are determined in each case. In order to accommodate Fermions using a first quantized Hamiltonian, an efficient quantum algorithm for anti-symmetrization is given. A simulation of the Hubbard model is discussed in detail, as well as a problem from quantum chemistry. I find that classically intractable and interesting problems from atomic physics could be solved with between 50 and 100 quantum bits.

A. Introduction

Since the discovery by Shor of a quantum algorithm for factoring in polynomial time [86], there has been tremendous activity in quantum computing. Recent results, some of which were discussed in the previous chap-

[24]The work described in this chapter is based upon [1] and [3].

ter, include the first experimental demonstrations of working quantum logic gates [75][92], quantum error-correcting codes[29][28][91][82], and many novel proposals for the design of actual quantum computers [31][67][90][7][77]. Despite these advances, however, the technical hurdles that stand in the way of factoring a large number on a quantum computer remain daunting [94][78][63]. On the other hand, the problem of simulation - that is, the problem of modeling the full time evolution of an arbitrary quantum system - is less technologically demanding. While thousands of qubits and billions of quantum logic operations are needed to solve classically difficult factoring problems[9], it would be possible to use a quantum computer with only a few tens of qubits and perhaps a few thousand operations to perform simulations that would be classically intractable. A quantum computer of this scale appears to be a more realistic possibility.

Because the size of the Hilbert space grows exponentially with the number of particles, a full quantum simulation demands exponential resources on a classical computer[25]. A system of only 100 spin 1/2 particles, for example, requires 2^{100} complex numbers to merely describe a general spin state. It is clear that on a classical computer, a simulation of this system is in general intractable. This exponential explosion severely limits our ability to perform true "ab initio" (first principles) calculations; since it is obviously not possible to even describe the state of anything but the smallest quantum systems, one must resort to various approximation techniques to calculate the properties of interest. The idea that a quantum computer might be more efficient than a classical computer at simulating real quantum systems was first proposed by Feynman (long before Shor's algorithm), but he speculated that the problem of Fermi statistics might prevent the design of a universal quantum simulator [46]. This chapter will deal explicitly with the problem of Fermions, in part by describing a quantum algorithm for antisymmetrization which executes in polynomial time. More recently, Lloyd has shown how a quantum computer is in fact an efficient quantum simulator [68]. In this chapter, I shall provide the first detailed algorithms for a quantum simulation, using the Hubbard model in both first and second quantized formalisms. To emphasize the algorithmic aspect of this work, many subroutines executed by the quantum computer will be described with pseudo-code instructions as well as with words.

Other recent work in quantum computation has revealed various techniques for *simulating* physics on a quantum computer [68][1][23][102][99][64]. However, while other work has described a variety of algorithms for time evolving a quantum state [68][1][23] [102][99], there has been comparatively little work done on algorithms which *calculate static properties* of a physical system [64][102]. In particular, of all the questions which one might ask about a quantum system, there is one most frequently asked and for which one would most greatly desire an efficient algorithm: What are the energy eigenvalues and eigenstates? This chapter will provide a quantum algorithm that can find eigenvalues and eigenvectors of a Hamiltonian operator in cases that occur frequently in problems of physical interest. Moreover, the algorithm requires an amount of time which scales as a polynomial function of the number of particles and the desired accuracy, whereas all classical algorithms (with known complexity[26]) require an exponential amount of time.

Hence, this chapter provides for the first time a complete and detailed quantum algorithm for simulating and calculating the properties of a system of physical interest, and describes also the first and only other known, well-defined algorithm, other than the Shor algorithm and certain artificial problems constructed explicitly for this purpose [21][88], that is thought to gain an exponential speed increase by exploiting quantum computation. This chapter also attempts to make a careful estimate of the quantum resources that are required to solve a classically intractable problem, and finds that only 50-100 qubits are necessary. This estimate is more than an order of magnitude smaller than previous estimates of the qubits required to factor "interesting" numbers.[27]

B. The fermion problem and the Hubbard model

This section will discuss how one performs a quantum simulation of a many-body Fermi system, using the Hubbard model as a concrete example. The algorithm used to perform the simulation could be implemented on a variety of possible hardware schemes: the actual implementation of the quantum computer is not relevant, as long as it supports universal quantum computation[6][39][43][69]. (However, different physical implementations may of course be better or worse suited for different problems).

B.1 Description of the system

The problem considered here consists of n particles, each of which can be in any of m single particle states, labeled $1. .m$. These states might be sites in a lattice, or atomic orbitals, or plane waves, etc. The mapping of the model system onto the qubits of the computer depends on whether we choose a first or second quantized description. In many respects, the second quantized form appears naturally well-suited for quantum computation of Fermi systems: the occupation of each state must be either 0 or 1, which maps directly to the state of a qubit. In this case, the memory needed to map the state of the entire n particle system is

[25] At least, it is believed to require exponential resources. If a method were found to perform classical simulations of quantum mechanics without an exponential overhead, than all quantum algorithms could be reduced to classical algorithms — implying, among other things, a polynomial time *classical* factoring algorithm. In this sense, the problem of performing a quantum simulation is at least as hard as any problem in BQP. (However, because of a mathematical technicality, which there is no space to discuss here, it cannot be called BQP complete).

[26] It is possible that some Quantum Monte Carlo methods may scale polynomially for certain problems, but the scaling is not known. Moreover, these techniques typically have additional difficulties with Fermi systems and with excited energy eigenstates.

[27] Because of the obvious difficulties in constructing a quantum computer and maintaining coherence, it is important to look for problems which require as few qubits as possible.

m qubits (independent of n).[28] To treat a first quantized Hamiltonian, one may imagine a quantum word, or qu-word, as a string of qubits of length $\log_2 m$; one qu-word represents any integer in the range $1..m$, and, consequently, the state of one particle. The state of the entire physical system being simulated can therefore be represented by n qu-words, or $n \log_2 m$ bits. If the number of particles is much smaller than the number of possible states, a first quantized representation may be vastly more efficient ($n \log_2 m$ qubits vs. m qubits). In either representation, if the simulated physical system is in a superposition of many direct product states (as it is in general), then the quantum computer will be in a corresponding superposition of states in order to represent the correct physical state of the simulated system.

B.2 Fermions in the second quantized formalism

The problem of Fermions is handled more easily in the second quantized form because the statistics are incorporated into the raising and lowering operators. As usual, the calculation begins with all qubits in the $|0\rangle$ state. One can prepare the system in any state as long as it can be reached from the zero state using a relatively small number of quantum logic operations (that is, polynomial in m). Examples of such states include those in which the n particles are localized in individual lattices sites, momentum eigenstates, thermal states of non-interacting particles, and states in which particles obey k-particle correlations or entanglements (for small k). Thus Fermi statistics do not pose any additional complications for system preparation in the second quantized formalism. Because the statistics are incorporated into the raising and lowering operators, the additional complications occur during time-evolution.

In the Hubbard model, electrons move about a lattice of sites. Each site may be empty or occupied by a spin up electron, a spin down electron, or two electrons of opposite spin. Two qubits are therefore required to represent the four possible states of each site. The Hamiltonian for the system is

$$H = \sum_{i=1}^{m} V_0 n_{i\uparrow} n_{i\downarrow} + \sum_{<i,j>\sigma} t_0 c_{i\sigma}^{\dagger} c_{j\sigma} \qquad (37)$$

In the first term, which corresponds to potential energy, V_0 is the strength of the potential, and $n_{i\sigma}$ is the operator for the number of Fermions of spin σ at site i. In the second term, which corresponds to kinetic energy, the sum $< i, j >$ indicates all neighboring pairs of sites, t_0 is the strength of the "hopping", and $c_{i\sigma}^{\dagger}$ and $c_{j\sigma}$are operators for creation and annihilation, respectively, of a Fermion at site i and with spin σ. The computer simulates the Hubbard model by performing the unitary operation $U = e^{-\frac{i}{\hbar}Ht}$ on suitably encoded states. This can be accomplished most easily by splitting the Hamiltonian into a sum of local terms H_j and repeatedly applying the operators $U_j = e^{-\frac{i}{\hbar}H_j \frac{t}{n}}$, to evolve local parts of the system over small time slices $\frac{t}{n}$, in series. (See Ref. [68] for further discussion of this technique; it will also be described in more detail below). Thus it suffices to describe algorithms which perform the time-evolution corresponding to each local term in the Hamiltonian. To effect the time evolution corresponding to the potential energy terms $\sum_{i=1}^{m} V_0 n_{i\uparrow} n_{i\downarrow}$, the following simple algorithm will suffice: consider each site one at a time; for each site, if it is occupied by two electrons (of opposite spin), advance the phase of the entire state by $-\frac{i}{\hbar} V_0 \frac{t}{n}$. This subroutine requires $O(m)$ operations. It is often convenient to describe a quantum algorithm in terms analogous to the pseudo codes used to describe classical algorithms; the previous algorithm then appears as follows:

```
loop i over sites
    if site i is occupied by two electrons,
        set flag qubit
        use controlled rotation to advance phase
of flagged components
    undo if (to restore the flag qubit)
```

To calculate the effect of the hopping terms $\sum_{<i,j>\sigma} t_0 c_{i\sigma}^{\dagger} c_{j\sigma}$ requires a slightly more complicated algorithm: first, loop over all pairs of (physically) neighboring sites i and j, and consider hopping between each pair of sites separately. For each pair i, j, count the number of occupied states which fall between i and j when the system is written in second quantized form. A flag is set to the parity of this number, which indicates whether or not a change of sign is introduced when hopping between the two sites. It is now easy to perform the time evolution U_i because the action of the operator takes place in the two qubit space i, j; a simple two qubit unitary operation is performed that diagonalizes the Hamiltonian in this space (depending upon the flag), and the phases of the eigenstates are then advanced by the appropriate amount. In pseudo code, the algorithm appears as follows:

```
loop i over states
    loop j over neighbors with i > j
        count states occupied between i and j
        let flag qubit = parity of this number
        diagonalize bits i,j according to parity
bit
            advance phase
        undo diagonalize
        undo counting and restore flag qubit
```

Assuming that the number of neighbors is a constant, the loops execute $O(m)$ times. It takes no more than $O(m)$ operations to count the occupancy of the intervening states, and it follows that the entire algorithm for simulating the second quantized Hubbard model executes in $O(m^2)$ quantum logic operations.

B.3 Fermions in the first quantized formalism

Fermi statistics are more difficult to handle in the usual first quantized description, because it is necessary to initialize the quantum computer into an antisymmetrized superposition of states corresponding directly to the actual physical state of the system. As there are $n!$ states in the

[28] For Bose particles, where the occupation number of each state can be any integer in the range $0..n$, $\log n$ qubits are needed to represent each state, yielding a total of $m \log n$ qubits.

superposition, one needs a fast quantum algorithm for generating this superposition state in order for the approach to be tractable.

The algorithm described below accepts as input a string of n qu-words (representing the state of the physical system being modeled) and generates an antisymmetrized superposition of $n!$ states in $O(n^2)$ time[29]. Note that without further restriction, antisymmetrization is an irreversible process and cannot be performed by a reversible quantum computer: there are $n!$ input states which correspond to the same antisymmetrized state (modulo an overall phase). One must therefore add the additional requirement that the input state is ordered. The correspondence between an ordered n-tuple of qu-words and an antisymmetrized superposition is one to one. In fact, this observation is in some sense the key to the algorithm.

System preparation in the first quantized formalism therefore begins by first initalizing the computer into an unsymmetrized state and then antisymmetrizing that state. Placing the system in any (unsymmetrized) direct product state is easy: simply place each particle in the appropriate single particle state. These single particle states might include, for example, those which are localized in position space, momentum space (obtained by using a quantum FFT), and thermal states. The system could also be initialized into states with arbitrary k-particle correlations or entanglements by performing quantum logic operations in the appropriate k-particle space, requiring only $O(m^{2k})$ operations in the general case, and often far fewer.

Antisymmetrization is accomplished in four main steps, summarized below:

Step I. Initialization of the input state. Imagine that there is a string of qubits which are all initially set to zero, and define three registers A, B, and C, each consisting of n qu-words ($n \log m$ qubits). The qubits in register A are initialized to the ordered string of qu-words which represent the input state of the system . The algorithm is unaffected if this state is a superposition of several ordered n-tuples.

Step II. Generating $n!$ states. We begin by creating the following superposition of states in register B:

$$\frac{1}{\sqrt{n}} \sum_{i=1}^{n} |i\rangle \otimes \frac{1}{\sqrt{n-1}} \sum_{i=1}^{n-1} |i\rangle \otimes \frac{1}{\sqrt{n-2}} \sum_{i=1}^{n-2} |i\rangle \otimes \\ ... \otimes \frac{1}{\sqrt{2}} (|2\rangle + |1\rangle) \otimes |1\rangle \tag{38}$$

This is accomplished with $O(n(\log m)^2)$ steps: by performing appropriate rotations on each qubit, one at a time, the computer is placed in a superposition of $n!$ unique states. For example, the sum $\frac{1}{\sqrt{8}}(|0\rangle + |1\rangle + ... + |6\rangle + |7\rangle)$ is easily generated by rotating three qubits from the state $|0\rangle$ into the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. States of the form $\frac{1}{\sqrt{j}} \sum_{i=1}^{j} |i\rangle$ for values of j that are not powers of two can be generated in a similar manner, one qubit at a time, by using controlled rotations that are conditioned on previous qubits.

Step III. Transform into permutations of natural numbers. The goal of this third step is to transform the superposition of states in register B into the superposition $\frac{1}{\sqrt{n!}} \sum_{\sigma \in S_n} |\sigma(1..n)\rangle$, where S_n is the symmetric group of permutations on n objects. This is an equal superposition of the states representing all the permutations of the first n natural numbers. The basic idea is as follows: let $B[i]$ indicate the i^{th} qu-word in register B; map $B[i]$ into a qu-word $B'[i]$ by setting $B'[i]$ equal to the $B[i]^{th}$ natural number less than n which does not occur in $B'[1]...B'[i-1]$. For example, the state $|1111\rangle$ maps into the state $|1234\rangle$; the state $|3321\rangle$ maps to the state $|3421\rangle$. This transformation is accomplished as follows:

```
loop i over qu-words 2..n
    Sort B[1].  .  .  B[i-1]
        {Note:  this will generate work bits for
reversibility}
    loop j from 1 to i-1
        if B[j]<=B[i] then increment B[i]
    end loop j
    Undo Sort {this will also cleanup the work bits}
end loop i
```

The algorithm described above requires only $O(n^2)$ operations (up to polylogarithmic factors). To prepare for the last phase of the algorithm the n-tuple $1, 2, 3...n$ is then assigned to register C, leaving the computer in the state:

$$\frac{1}{\sqrt{n!}} |\Psi\rangle \otimes \left(\sum_{\sigma \in S_n} |\sigma(1..n)\rangle \right) \otimes |1..n\rangle \tag{39}$$

Step IV. Sorting and unsorting. The algorithm proceeds with a series of sorting and unsorting operations. As in Step III, a string of work qubits is required so that the sorting operations are reversible. Any sorting algorithm can be used; however, a heap sort is recommended, because it requires $O(n \log n)$ operations in all cases and only $n \log n$ scratch qubits.[30] The first sort orders register B with a series of exchanges and scrambles A and C with the same series of exchanges. The resulting state is

$$\frac{1}{n!} \sum_{\sigma \in S_n} |\sigma(\Psi)\rangle |1..n\rangle |\sigma(1..n)\rangle |scratch\rangle \tag{40}$$

At this point, one has already obtained a symmetrized superposition of the input states, but it is entangled with many other qubits. One can antisymmetrize simply by counting the number of exchanges made during the sorting operation and advancing the phase of that component of

[29]The complexities in this section are often specified only up to a poly-logarithmic factor. For example, the entire algorithm is more accurately described as $O(n^2(\log m)^2)$.

[30]Note that in a quantum algorithm such as the one described here, the complexity of the sorting is the *worst-case* complexity, because one cannot look at the state of the quantum computer to determine if the sorting has been completed, and because there are undoubtedly some elements in the superposition which will take the worst-case time. Thus a QuickSort, which requires $O(n \log n)$ operations on average, would require $O(n^2)$ operations in the quantum implementation.

the superposition by π (i.e., flip the sign) if this number is odd. (If one wishes to obtain a symmetrized state for a simulation of Bose particles, one can simply leave the state as it is and proceed). The algorithm continues by reversing the sort on register B, but leaving registers A and C unchanged. The qubits contained in B and C are then redundant: in each component of the superposition, if B[i] = n, then C[n] = i. This redundancy allows B to be set to zero reversibly. By then sorting A and C together, eliminating C, and unsorting, one obtains the desired antisymmetrized state. Note that in the final unsorting operation, the algorithm relies upon the fact that the ordering of the input state $|\Psi\rangle$ was stipulated to be the same as the ordering of the integers 1..n in register C (so that antisymmetrization would be reversible); if this were not the case, the algorithm would fail. The entire process is completed in $O(n^2)$ operations.

Because the input state is now fully antisymmetrized, time evolution is in principle straightforward. Using the same technique as before, the Hamiltonian is split into a sum of terms H_j and the corresponding time evolution operators $U_j = e^{-\frac{i}{\hbar}H_j\frac{t}{n}}$ are applied to the state in series. (The antisymmetry of the state will not be effected by time step errors that occur during this process; although each individual U_j does not preserve antisymmetry, their products do exactly.) Each U_j can be performed by an appropriate series of quantum logic operations; the actual sequence of gates required can be determined by inverting the Campbell-Baker-Hausdorff formula. Using this procedure, $O(m^2)$ steps are required to perform an arbitrary one particle operator U_j, and $O(m^4)$ operations are required to perform an arbitrary two particle operator. It is therefore possible to simulate in polynomial time any system of Fermions using the first quantized description. For the special case of the Hubbard model, the simplicity of the Hamiltonian allows one to do better. I describe below how to perform each U_j in only $O((\log m)^2)$ steps. To begin, consider the Hubbard model Hamiltonian in its first quantized form:

$$H = \sum_{i=1}^{n} T_i + \frac{1}{2}\sum_{k\neq l} V_{kl} \qquad (41)$$

where $\langle i\sigma|T|j\sigma\rangle = t_0\delta_{<i,j>}$, and $\langle i\uparrow, i\downarrow|V|i\uparrow, i\downarrow\rangle = V_0$ are the only nonzero matrix elements of V. As before, the potential energy terms are easier because they are diagonal. For a given pair of particles, simply determine if they are at the same site and perform a controlled rotation if they are. In code:

```
loop i,j over pairs of particles
if i,j are at same site set flag qubit
        use controlled rotation to advance phase
of flagged components
     undo if (to restore the flag qubit)
```

In order to perform the time evolution corresponding to the kinetic energy terms, we focus on one particle at a time. For each particle, the idea is to decompose the kinetic energy terms into a sum of block diagonal matrices and then diagonalize the sub-blocks in each matrix in parallel.[31] For simplicity of explanation, consider a 1-d Hubbard model and ignore spin. The general case is a straightforward extension. In the 1-d spinless case the kinetic energy part of the Hamiltonian can be written

$$T = h(1,2) + h(2,3) + h(3,4) + ... + h(m-1,m) \qquad (42)$$

where $h(i,j)$ is the piece of the Hamiltonian that corresponds to hopping between sites i and j: all matrix elements of $h(i,j)$ are zero except $\langle i|h(i,j)|j\rangle = \langle j|h(i,j)|i\rangle = t_0$. One can rewrite the previous expression as follows:

$$T = T_1 + T_2 \qquad (43)$$

$$T_1 = h(1,2) + h(3,4) + h(5,6) + ... \qquad (44)$$

$$T_2 = h(2,3) + h(4,5) + h(6,7) + ... \qquad (45)$$

The operators T_1 and T_2 are in block diagonal form. To fully diagonalize each matrix (separately), perform quantum logic operations on each state to transform the state number into two quantum numbers labeling the block and the location within the block (0 or 1). For example, to diagonalize T_1, map the state $|n\rangle$ into $|(n+1) \text{ div } 2, n \bmod 2\rangle$ (where x div 2 indicates the greatest integer less than or equal to x/2). Because T_1 is block diagonal and only mixes states within each block - and because all states within the same block have their first quantum number in common - the action of T_1 takes place entirely within the space of the second quantum number. In this one qubit space it is simply the matrix $t_0\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Thus all the blocks can be diagonalized in parallel by diagonalizing this trivial 2x2 matrix in the one qubit space of the second quantum number. Each state in the superposition is then advanced by the appropriate phase, and all the previous steps are reversed. The algorithm requires only $O((\log m)^2)$ quantum logic operations.

B.4 Reading the final state

Having thus described how to prepare a quantum computer in a state that is analogous to the initial state of a many body Fermi system — and how it can be programmed to simulate that system's time evolution — one must now consider what information can be extracted from a quantum many-body simulation. It is obviously impossible to obtain the entire wavefunction (no classical memory could hold it). Rather, the "answer" is obtained by performing a series of measurements on the qubits, one at a time. Each such measurement will yield either a $|0\rangle$ or a $|1\rangle$. It is thus possible to measure any physical property of the wavefunction that can be expressed in terms of such local variables. To obtain useful information about the physics of the simulated system, one must initialize the quantum computer, simulate time-evolution, make a measurement, and then repeat this process a sufficient number of times to acquire a statistically significant result.

[31] Because one often encounters near-diagonal operators in various problems, this technique may likely be useful in other circumstances besides simulations.

One important example is the electronic charge density distribution. In the second quantized representation, one performs measurements at each site to determine the probability of occupancy. The number of such measurements required to obtain some desired accuracy ϵ varies as ϵ^{-2} (i.e., the accuracy grows as a polynomial function of the number of trials). In the first quantized representation, the same result is obtained by measuring the location of a given particle and generating a histogram of locations from repeated trails.

It is possible to obtain two-particle correlation functions and even k-particle correlations using a similar approach (requiring roughly $O(\epsilon^{-2}\delta^k)$ trials, where δ is the density of points in the histogram and ϵ is the desired accuracy). The momentum distribution function can be obtained by performing a quantum FFT before sampling the wavefunction. From the one and two particle densities and the momentum distribution, it is possible to calculate the expected energy of the wavefunction.

A variety of other techniques might be used to obtain other types of information: for example, one could obtain scattering amplitudes by simulating the motion of an electron through a charged medium and measuring the probability of its emerging with different momenta. Or, one might perform a quantum simulated annealing by time-evolving the system in contact with a simulated heat bath. By then employing the previous techniques one could obtain information about the system's ground state.

C. Finding eigenvalues and eigenvectors

Although the techniques described in the previous section allow one to initalize a simulation in a variety of physically interesting states, time evolve that system, and make measurements to learn the relevant physical information about the final state, it is not clear how one can determine the properties that are often of most interest: eigenvalues and eigenvectors. This section provides an algorithm for finding this information.

C.1 Statement of the problem

The problem to be solved will be formalized as follows. Consider the time-evolution operator $\widehat{U} = e^{-\frac{i}{\hbar}\widehat{H}t}$ which corresponds to the Hamiltonian $\widehat{H}$, and an approximate eigenvector V_a of $\widehat{U}$ (and thus of $\widehat{H}$) that can be generated in quantum polynomial time; i.e., the machine can be placed into a state corresponding to V_a using a polynomial number of quantum logic operations. Call the true eigenvector V and the true eigenvalue λ_v. If the state V_a satisfies the property that $|\langle V_a|V\rangle|^2$ is not exponentially small - that is, the approximate eigenvector contains a component of the actual eigenvector that is bounded by a polynomial function of the problem size - then V and λ_v can be found in time proportional to $1/|\langle V_a|V\rangle|^2$ and $1/\epsilon$, where ϵ is the desired accuracy.

Intuitively, what the algorithm does is to resolve the guess into its non-negligible components and determine the corresponding eigenvalues. If the operator $\widehat{U}$ (and thus its eigenvectors) is of exponentially large dimension - which it typically is - there are no known classical algorithms that can find even the eigenvalues in polynomial time. Although the requirement that there exist an initial state vector V_a with the specified properties may appear to be overly restrictive, it is frequently (if not usually) possible to obtain such a guess for "real" problems using existing classical techniques. For example, in any physical system with discrete energy levels that are not exponentially close together near the ground state (such as an atom), if it is possible to obtain classically any state vector with expected energy merely less than the first excited state (by a non-exponentially small amount), then this state vector must contain a non-negligible component of the ground state and - although it may not remotely resemble the ground state - could be used as the approximate state V_a to determine the true ground state and ground state energy in polynomial time. In fact, for a system with discrete energy levels, almost any physically reasonable initial state is likely to contain non-negligible components of all nearby eigenvectors (which this algorithm would therefore find). Finally, if for some problems it is not possible to obtain classically an approximate state with the desired properties, it may often be the case that the state vector V_a may be generated using a quantum algorithm, such as quantum simulated annealing.

C.2 Eigenvalues and eigenvectors via quantum FFT

The core of the quantum algorithm is a subroutine which applies to any $\widehat{U}$ that can be implemented in quantum polynomial time. (It was shown in [68] that the time evolution operator corresponding to any local Hamiltonian can be implemented in polynomial time on a quantum computer.) (As an example, the Hubbard model, as discussed in the previous section, would be one possible $\widehat{U}$ for which the necessary quantum logic gates are known in detail). A similar subroutine was previously (though independently) described by Cleve et. al. in [37] to find the eigenvalues of unitary operators. Cleve et. al. show how one can obtain an eigenvalue with *exponential precision* if one is initially given the eigenvector and devices that can perform $\widehat{U}$,$\widehat{U}^2$,$\widehat{U}^4$, ... ,$\widehat{U}^{2^m}$; they use this subroutine in a modified version of Shor's algorithm by randomly sampling from the eigenspectrum. Of course, we have no way to generate $\widehat{U}^{2^m}$, nor do we seek random eigenvalues or exponential precision (polynomial accuracy will suffice). However, we employ essentially the same subroutine in an algorithm that takes an estimate of an eigenvector (e.g., the ground state) and uses it to find the eigenstate itself. (In its original form, this routine is not useful for finding eigenvectors: in a Hilbert space of exponential dimension the chances of getting the same eigenvector twice are exponentially tiny. The situation is similar to Shor's use of the quantum Fourier transform: the quantum Fourier transform, developed by Coppersmith[33], does not in and of itself supply an exponential speed up over classical computation. It is only when used as a subroutine in a quantum algorithm for factoring that it allows an exponential speed up over all known classical algorithms).

The algorithm will proceed as follows: Consider a quantum computer consisting of $m+l+w$ qubits, where a total of m qubits (to be called the index bits) are used for an FFT, a total of l qubits describe the Hilbert space in which the operator $\widehat{U}$ acts, and w extra working qubits are required for temporary storage. Let $M = 2^m$. The accuracy of the result will grow as $1/M$ — therefore, the required number of qubits will scale as the log of the accuracy. Assume that the m index qubits are initially in the state $|0\rangle$ and that the l qubits are initially in the state V_a; i.e., the initial state is

$$|\Psi >= |0 > \ |V_a > \tag{46}$$

where the w work qubits are assumed to be $|0\rangle$ unless specified otherwise. Perform a $\pi/2$ rotation (Walsh-Hadamard transform) on each of the m index qubits to obtain the state

$$|\Psi >= \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |j\rangle |V_a\rangle \tag{47}$$

Next, one performs a series of quantum logic operations that transform the computer into the state

$$|\Psi >= \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |j\rangle (\widehat{U})^j |V_a\rangle \tag{48}$$

This transformation is accomplished by applying the operation $\widehat{U}$ to the second set of l qubits (which are initially in the state $|V_a\rangle$) a total of j times. It can be implemented easily by performing a loop (indexed by i) from 1 to M. Using standard quantum logic operations, set a flag qubit to the value $|1\rangle$ if and only if i <j and perform the operation $\widehat{U}$ conditioned on the value of this flag. Thus only those components of the above superposition for which i<j are effected. Finally, undo the flag qubit and continue with the next iteration. After M iterations, the state above is obtained.

At this point, it is helpful to rewrite the state in a slightly different manner. Label the eigenvectors of $\widehat{U}$ by the states $|\phi_k\rangle$ and the corresponding eigenvalues with λ_k. One can then write

$$|V_a\rangle = \sum_k c_k |\phi_k\rangle \tag{49}$$

in which case the state (48) above can be rewritten as

$$|\Psi\rangle = \frac{1}{\sqrt{M}} \sum_{j=0}^{M-1} |j\rangle (\widehat{U})^j \sum_k c_k |\phi_k\rangle \tag{50}$$

$$= \frac{1}{\sqrt{M}} \sum_k c_k \sum_{j=0}^{M-1} |j\rangle (\lambda_k)^j |\phi_k\rangle \tag{51}$$

If we write λ_k as $e^{i\omega_k}$ and exchange the order of the qubits so that the labels $|\phi_k\rangle$ appear first, the result is seen then most clearly:

$$|\Psi\rangle = \frac{1}{\sqrt{M}} \sum_k c_k |\phi_k\rangle \sum_{j=0}^{M-1} e^{i\omega_k j} |j\rangle \tag{52}$$

It is now self-evident that a quantum FFT performed on the m index qubits will reveal the phases ω_k and thereby the eigenvalues λ_k. The quantum FFT requires only poly(m) operations, whereas the accuracy of the result will scale linearly with $M = 2^m$. Each frequency is seen to occur with amplitude $c_k = \langle V_a|\phi_k\rangle$; by performing a measurement on the m index qubits, one thus obtains each eigenvalue with probability $|c_k|^2$. Only a polynomial number of trials is therefore required to obtain any eigenvalue for which c_k is not exponentially small. If the initial guess $|V_a\rangle$ is close to the desired state (i.e., $|< V_a|V >|^2$ is close to 1), then only a few trials may be necessary.

Moreover, once a measurement is made and an eigenvalue λ_k is determined, the remaining l qubits "collapse" into the state of the corresponding eigenvector. One is likely to be interested in various properties of the eigenvectors, and these can be determined by making various measurements on the state, as described in the previous section. For *ab initio* quantum calculations, easily obtainable properties include those of greatest interest: charge density distributions, correlation functions, momentum distributions, etc. Of course, the state $|\phi_k\rangle$ is still in some sense "trapped" inside the computer. But since it is impossible to store as classical information the 2^l phases associated with the state, one cannot possibly do better. Nevertheless, the relevant physical information can be extracted efficiently from the quantum computer.

An interesting subtlety occurs if the eigenvalue found above is degenerate or nearly degenerate (that is, there are several eigenvalues which differ by less than the accuracy $1/M$). (Note however that nearly degenerate states can be resolved in polynomial time, if desired, as long as they are not exponentially close together.) For degenerate or nearly-degenerate eigenvalues, the measurement projects the system into the corresponding subspace. One can then determine properties of this subspace - that is, the relevant physical properties of the system - through additional measurements as described above. However, one can also use this technique to detect the presence of a degeneracy by simulating a small perturbation or by varying the initial conditions.

C.3 Applying the algorithm

Let us now consider more precisely how to use this subroutine to find the eigenvectors and eigenvalues of a "real" Hamiltonian. Generally, one wishes to find energy eigenstates for a Hamiltonian of the form

$$H = \sum_{i=1}^{n} (T_i + V_i) + \sum_{i>j}^{n} V_{ij} \tag{53}$$

where n is the number of particles, T_i is the kinetic energy, V_i is the external potential, and V_{ij} is the interaction between the particles. (Other terms can be included, as long as they act locally). Because the Hamiltonian is Hermitian, one applies the steps described above to the time evolution operator $\widehat{U}(t) = e^{-iHt}$, which is unitary and has the same

eigenvalues and eigenvectors. This time evolution operator is generated using the same technique as in the previous section (see also [68]); the key idea is to write $H = \sum H_i$ (where each H_i acts on only k qubits at a time) and

$$\widehat{U}(t) = e^{-iHt} = (e^{-iH_1\frac{t}{m}}e^{-iH_2\frac{t}{m}}...e^{-iH_k\frac{t}{m}})^m + \sum_{i>j}[H_i, H_j]\frac{t^2}{2m} + ... \tag{54}$$

Let $U_i = e^{-iH_i\frac{t}{m}}$. Each term U_i can be implemented efficiently, because it acts in a space of only k quantum bits, where k is small. For large enough m, the second term on the right (and the higher order terms) approaches zero. It is therefore possible to generate $\widehat{U}(t)$ by acting on the state with each U_i in series, a total of m times. In order to simulate $\widehat{U}(t)$ with an accuracy ϵ, one needs to apply $O(t^2/\epsilon)$ quantum logic operations.[32]

For a specific problem, the form of the matrices U_i depends greatly on the basis set chosen to describe the Hilbert space. Moreover, the choice may strongly impact the size of the basis required to describe the system accurately. Virtually any basis set may be used: position space, momentum space, wavelets, single electron solutions for an effective potential, etc. As long as the single particle basis is of a fixed size, then the operators U_i can always be calculated in the chosen basis and implemented using $O(d^4)$ operations, where d is the dimension of the *single particle* basis set [6]. On the other hand, there is a trade-off between memory and speed. By using the position or momentum space representation, one needs only O(poly(k)) = O(poly(log d)) operations to perform each U_i; however a large number of qubits are required to describe the eigenstates accurately. By choosing a more elaborate basis set, one can vastly reduce the required number of qubits, but a much larger number of quantum logic operations $O(d^4)$ may be necessary to implement each U_i. (The trade-off described here is similar to the trade-off between first and second quantized representations discussed in the previous section). Thus one finds that, just as with conventional computations, the choice of basis sets in the quantum computation will depend upon the specific problem at hand and the specific capabilities of the actual computing machine.

Normally, the initial state V_a will be the result of a classical calculation, for example, a Hartree-Fock calculation or configuration interaction calculation. Any *ab initio* technique which results in a known wave function can be used. (Note that this does not include those techniques which utilize density functional theory, as we require a wavefunction, not simply a charge density distribution). If the input wave function is not already symmetrized or antisymmetrized, one may use the algorithms described in the previous section to do so efficiently.

Finally, let us consider a state-of-the-art ab initio calculation of atomic energy levels in order to compare the quantum algorithm described above with known classical techniques. Problems from atomic physics serve as a particularly good benchmark because extremely accurate experimental data is widely available. The quantum algorithm corresponds most closely to what is known as "complete active configuration interaction" or "full configuration interaction" techniques, because the many-particle basis set includes all possible products of single particle basis vectors. This approach is most valuable in situations where the correlation energy is large and where many "configurations" are of similar energy (this typically occurs when many electrons are in open shells). Unfortunately, it is difficult to state precisely the minimum size problem for which the quantum calculation surpasses the best classical calculations, because a variety of sophisticated techniques are used to avoid the exponential explosion in basis states. That is, the most *accurate* classical calculations do not employ directly the "full configuration interaction" method. Based on [58], however, it appears that a calculation of the energy levels of B (5 electrons), using roughly 20 angular wavefunctions and 40 radial wavefunctions per particle - for a total of 800 single particle wave functions and therefore $800^5 \approx 10^{15}$ full many-body basis states - may provide more accurate results than any classical calculation performed to date. At the very least, such a calculation would reveal scientifically interesting (and classically unobtainable) results with respect to electron correlation energies in B and the relative importance of various orders of excited configurations.

A quantum calculation of the B ground state, using a basis set as described above, can be accomplished with 60 qubits: 10 per particle (allowing 1024 states in the single-particle basis) to represent the state of the atom (for a total of 50 qubits), 6 or 7 qubits for the FFT, and a few additional "scratch" qubits[33]. Unfortunately, the two particle operators (generated by the coulomb attraction between pairs of electrons) take place in a subspace of dimension $(2^{10})^2$; they therefore are represented by matrices with 2^{40} elements. Implementing such an operator by brute force is likely to remain intractable for the foreseeable future. However, it is possible to perform the necessary transformation using a quantum algorithm. One possible technique is to change basis sets: by representing the interacting particles in position space, instead of with the orbital basis set, it is

[32]Since U(t) has the same eigenvalues and vectors for all t, this might lead one to falsely conclude that the number of operations necessary to find the eigenstates to a given accuracy could be reduced by choosing a shorter length of time t for the operator U(t). However, the algorithm requires one to calculate U^M, and since $U(t)^M = U(Mt)$, one sees that U = U(t) must be calculated with greater precision if U^M is to be calculated for a fixed precision. In fact, since the eigenvectors are determined with a precision proportional to M, the number of quantum logic operations required to calculate the energy eigenstates to a precision ϵ is seen to scale as ϵ^{-2}.

[33]The number of qubits required for the FFT is not as large as one might at first suppose, based on the earlier statement that the accuracy scales linearly with the size of the FFT. This statement is true only for a fixed U. By changing U - in particular, by increasing the length of time t in U(t) - one can obtain the eigenvalues to arbitrary precision using a fixed number of FFT points. However, the number of points in the FFT must be sufficiently large so as to seperate the frequencies corresponding to distinct eigenvectors. This is how the estimate of 6 or 7 qubits (64 or 128 FFT points) is made.

easy to calculate the coulomb terms (because they are diagonal in this basis). Thus one can transform each particle into position space separately (requiring a small number of quantum logic operations), perform the time evolution corresponding to the coulomb interaction, and then transform back. Unfortunately, a position space representation will require many more qubits. A fairly conservative estimate is that 30 qubits per particle (10 per dimension, for a real space grid of 1024x1024x1024 per particle) will more than suffice. Because these 30 qubits are required only temporarily for the 2 particles whose interaction we are considering at any particular stage in the algorithm, the new efficient algorithm requires a total of 2 x 30 qubits (for the interacting particles), an additional 3 x 10 qubits (for the remaining particles), and the same 10 qubits for the FFT and work space. It thus appears that in order to realistically perform an "interesting" calculation using the algorithms described previously, one will need a quantum computer with approximately 100 qubits. Of course, the possibility remains that an efficient algorithm for implementing the coulomb interaction could be invented that does not require additional working space.

D. Conclusion

In summary, I have explicitly demonstrated how a universal quantum computer can be used to efficiently simulate systems consisting of many Fermions. Depending on the particular problem, it may be preferable to employ second quantized notation (requiring m qubits) or first quantized notation (requiring $n \log m$ qubits). An $O(n^2)$ algorithm for creating an antisymmetrized superposition of states has been described. This chapter also provided detailed algorithms which will simulate the Hubbard model, requiring $O(n^2)$ quantum logic operations in first quantized form, and $O(m^2)$ operations in second. The former algorithm employs a scheme for accommodating nearly diagonal Hamiltonians that might be applied to a wider range of problems.

Finally, this chapter demonstrated a new quantum algorithm which can be used to find eigenvectors and eigenvalues of a Hamiltonian operator. The algorithm provides an exponential speed increase when compared to the best known classical techniques. Problems from atomic physics may be the best place to perform the first real calculations, both because accurate experimental data is available to verify the resulting calculations, and because the parameters involved appear to be within the foreseeable range of small quantum computers. I have estimated that 50 - 100 qubits would be sufficient to perform "interesting" calculations that are classically intractable.

IV. Integrals and Stochastic Processes

[34]**Summary.** This chapter will discuss quantum algorithms that calculate numerical integrals and various characteristics of stochastic processes, and describe how one may apply either quantum counting or Grover's mean estimation algorithm to solve these problems. Both of these techniques obtain an exponential speed increase in comparison to the fastest known classical deterministic algorithms and a quadratic speed increase in comparison to classical Monte Carlo (probabilistic) methods. I derive a simpler and slightly faster version of Grover's mean algorithm, show how to apply quantum counting to the problem, develop some variations of these algorithms, and show how both (apparently distinct) approaches can be understood from the same unified framework. Finally, I'll discuss how the exponential speed increase appears to (but does not) violate results obtained via the method of polynomials, from which it is known that a bounded-error quantum algorithm for computing a total function can be only polynomially more efficient than the fastest deterministic classical algorithm.

[34]This chapter is based upon work [5] which took place while the author was visiting NASA-JPL and which will be published separately.

A. Introduction

Although quantum algorithms have been discovered that can solve many problems faster than the best known classical algorithms, there is a general sense — due, perhaps, to the enormous technical challenges that must be overcome before a useful quantum computer can ever be built — that more applications must be found in order to justify attempts to construct a quantum computing device.

In this chapter I suggest one possible application of a quantum computer, namely, computing the values of integrals. This problem can be solved in a fairly straightforward manner via either quantum counting[25], or Grover's mean estimation algorithm[57]. Although these general algorithms are not new, this application may be the most useful one described to date. (Because N operations are required to retrieve N values from a classical database, the mean finding algorithm affords no speed-up when applied to a pre-existing data set. Indeed, even the original database search algorithm has only limited utility, because it can only be used to search a function space. It is not clear to how many real-life problems it could be applied [103]).

I also suggest that a quantum computer may be used to determine various characteristics of stochastic processes. Frequently, such processes are used to generation distribution functions, and one wishes to know the mean, variance, and higher moments. One can apply quantum counting and mean estimation to obtain super-classical speedups for these problems as well.

On a quantum computer, one can find the value of a d-dimensional integral in $O(1/\epsilon)$ operations, where ϵ is the desired accuracy. It follows from the results of Nayak and Wu [76] that this is in fact a lower bound. Classically, one requires $O(1/\epsilon^2)$ operations to achieve the same accuracy using probabilistic methods, and requires $O(1/\epsilon^d)$ - *exponentially* more - operations to achieve the same accuracy deterministically. (More precisely, it is polynomial in the accuracy and exponential in the number of dimensions.) Since real computers and all classical devices are

in fact deterministic, this exponential speed increase is by no means a red herring. Indeed, there is a popular misconception that real computers can perform probabilistic algorithms with impunity by employing pseudo-random number generators. Of course, pseudo-random numbers are not truly random at all - and one must in fact be careful about treating them as such. For example, in 1992 Ferrenberg et al. found bugs in a supposedly good pseudo-random number generator when a numerical simulation of an Ising spin system failed due to hidden correlations in the "random" numbers[49]. The moral here is that one cannot rely upon a classical computing device to properly execute a probabilistic algorithm. In some sense, one could argue that the quantum algorithm for evaluating integrals provides an exponential speed increase.

B. Statement of the problem and classical algorithms

Without loss of generality, one may consider integrals of a real-valued d-dimensional function $g(x_1, x_2, ...x_d)$ defined for x_i in the range $[0, 1]$ and where $g(x_1, x_2, ...x_d) \in [0, 1]$, for all values of x_i. Thus one seeks to calculate

$$I = \int_0^1 \int_0^1 ... \int_0^1 g(x_1, x_2, ...x_d) dx_1 dx_2 ... dx_d \qquad (55)$$

In the discussion that follows, g will be approximated with a real-valued d-dimensional function $f(a_1, a_2, ...a_d)$ defined over integral values a_i in the range $[1, M]$ and where

$$f(a_1, a_2, ...a_d) = g(\frac{a_1}{M}, \frac{a_2}{M}, ... \frac{a_d}{M}) \qquad (56)$$

Thus, we wish to find the sum

$$S = \frac{1}{M^d} \sum_{a_1=1}^{M} \sum_{a_2=1}^{M} ... \sum_{a_d=1}^{M} f(a_1, a_2, ...a_d) \qquad (57)$$

Note that the sum S is identical to the average of f over all a_i. The accuracy with which the sum S approaches the integral I is obviously determined by the density of points M in each variable and the shape of the particular function. However[35], in what follows, our sole concern will be with approximating the sum S.

A sum of this form can also be used to determine properties of a stochastic process. A stochastic process may be described by a sequence of values, $w_1, w_2, ..., w_N$, where each value w_i is chosen randomly from a distribution which may depend on some (or all) w_j for $j < i$. For example, a simple random walk would be described by a sequence for which each w_i is either $(w_{i-1} + 1)$ or $(w_{i-1} - 1)$ with equal probability. Often, one is interested in a property of such a sequence that can be represented as a function $v(w_1, w_2, ..., w_N)$. (In many cases, the function v may depend only upon the final value w_N). One wishes to determine the mean, variance, skewness, and possibly higher moments of the function v over the space of all possible sequences. This problem is easily transformed into the form (55) through a change of variables: write each w_i as a function $w_i(r_i, w_1, w_2, ..., w_{i-1})$, where r_i is a random variable in the range [0,1]. Then one can write v as a function $v(r_1, r_2, ..., r_N)$ of the independent random variables r_i, scale the output so that it fits within the desired range, and obtain a function in the form g above. The mean value of the stochastic process is then simply the integral (55). Once again, the integral is represented as a discrete sum. (For some stochastic processes, the problem may in fact be discrete from the beginning). Thus the problem again reduces to finding the sum S in (57).

One can find higher moments of a stochastic process by simply applying the above approach to a calculation of the mean of v^2, v^3, etc. This method can of course also be applied to calculate moments of any distribution function (even if it is not the result of a stochastic process) as long as it can be represented in closed form.

It should be intuitively obvious that without any knowledge of the function f, one requires classically $O(M^d)$ operations to evaluate the sum. More precisely, if one views f as an oracle (or "black-box"), then one requires at least $M^d/2$ queries to determine S to within $\pm\frac{1}{4}$. (This is because it is possible that the remaining $M^d/2$ unqueried function values may be either all 0's or all 1's, one of which will always shift the mean by at least $\frac{1}{4}$). It follows that an ordinary classical Turing machine requires exponentially many operations (as a function of d) to determine S with accuracy ϵ for any $\epsilon < \frac{1}{4}$.

However, if one is allowed to employ a probabilistic algorithm, then one can randomly sample values of the function f for various $a_1, a_2, ...a_d$; as long as the values of a_i are chosen randomly (and provided that you are not exceedingly unlucky), it is possible to quickly approximate S to any desired precision. Indeed, it is a straightforward consequence of the central limit theorem that one can determine S with accuracy ϵ (with bounded probability) using only $O(1/\epsilon^2)$ operations. Note that the number of trials does not depend at all upon the size of the function's domain - as it did in the deterministic case - but only on the desired accuracy. This is in fact how Monte Carlo integrals are computed, and is essentially the only practical way to calculate integrals of functions with high dimensionality. (It is also why one need not be concerned with the approximation of the integral I with the sum S - one can make M essentially as large as one desires, paying only a logarithmic cost in computational complexity). Unfortunately, Monte Carlo integrals on classical devices require the use of a pseudo-random number generator, and as mentioned previously, there is no guarantee that one will obtain "good" random numbers. One obvious way to solve this dilemma would be to use a simple quantum event to produce a string of truly random numbers; but once one introduces quantum mechanics into the problem, one can find an even more effective solution.

[35] Because the computational complexity of the quantum algorithms (and also the classical Monte Carlo algorithms, for that matter) depend only logarithmically on M, this approximation is not a limiting factor (as long as the function is not pathological).

C. Review of Grover searching

Both of the quantum algorithms discussed in this chapter require a generalized version of Grover searching. The treatment below follows that of Grover [57]; similar ideas have also been described by Brassard et. al. [25] and various others.

All quantum algorithms consist of unitary operations applied in series. Any sequence of unitary operations can be viewed as a single unitary operator. Consider a particular unitary operator U which has amplitude U_{ts} between a starting state $|s\rangle$ and a target state $|t\rangle$. If the computer is initially in the state $|s\rangle$, then after one application of U the computer will be found in the state $|t\rangle$ with amplitude U_{ts}, and if the state of the computer is measured in the canonical basis, the probability of obtaining the state $|t\rangle$ will therefore be $|U_{ts}|^2$. We seek to *amplify* the amplitude of the state $|t\rangle$. (Increasing the amplitude of this state increases the chances that it will be found upon measurement and thereby allows for fast searching).

Amplitude amplification in it's simplest form requires the inversion operator I_x which inverts the phase of the state $|x\rangle$. We compose the unitary operators I and U to form the unitary operator G in the following way:.

$$G = -I_s U^{-1} I_t U \tag{58}$$

It can be easily verified that the operator G leaves invariant the subspace spanned by $|s\rangle$ and $U^{-1}|t\rangle$. In particular, one finds that

$$\begin{aligned} G\left(\alpha|s\rangle + \beta U^{-1}|t\rangle\right) = &\left\{(1-4\,|U_{ts}|^2)\alpha + 2U_{ts}\beta\right\}|s\rangle \\ &+\left\{-2U_{ts}^*\alpha + \beta\right\}U^{-1}|t\rangle \end{aligned} \tag{59}$$

which is approximately a rotation by $2\,|U_{ts}|$ radians. It follows that by applying $O(1/\,|U_{ts}|)$ iterations, one can obtain the state $U^{-1}|t\rangle$ with near certainty.

The original fast searching algorithm [55][56] applies the above steps with $U = W$, where W is the Walsh-Hadamard transform - that is, a $\pi/2$ rotation of each qubit. If the initial state $|s\rangle = |00...0\rangle$, then $|U_{ts}| = |W_{ts}| = 1/\sqrt{N}$ for all possible target states $|t\rangle$. The unitary operation I_t selectively inverts the phase of the actual target state $|t\rangle$ for which we are searching. After one application of W, the probability of measuring $|t\rangle$ would be only $\frac{1}{N}$, the same as one would obtain classically by guessing. However, it follows immediately from the above that the amplitude $|t\rangle$ can be amplified to nearly 1 by applying only $O(\sqrt{N})$ operations.

D. Integrals via amplitude amplification

To evaluate the sum S in (57), one can use the mean estimation algorithm described by Grover in [57]. I provide a simpler version of this algorithm (that is also faster, because it requires about half as many quantum logic operations). The algorithm works by refining a series of approximations. One can obtain an intuitive understanding of the approach by employing an analogy to classical coin-flipping. Consider a coin, which, when tossed, comes up heads with probability $p = S$. By the central limit theorem, one can determine S with accuracy ϵ using $O(1/\,\epsilon^2)$ trials. Let us choose $\epsilon = 0.1$ so that after some (fixed) number of trials N we have determined (with bounded probability) the first digit of p. [36] As a concrete example, let $S = 0.7468332$: the first set of trials would then reveal S to be approximately 0.7. Call this first estimate E_1. Imagine now that we are given a second coin, which, when tossed, comes up heads with probability $p_2 = \frac{S-0.7}{0.1}$. Just as with the first coin, tossing the coin $O(1/\,\epsilon^2)$ times determines p_2 with accuracy ϵ. Choosing the same ϵ and number of trials N as before, we obtain (with bounded probability) the first digit of p_2. But this is the second digit of S, and thus the current estimate is now $E_2 = 0.74$. We then undergo a third iteration in which $p_3 = \frac{S-0.74}{0.01}$ in order to determine the third digit of S. Continued iterations of this process, with more and more refined "coins", allow one to determine S with arbitrary accuracy, each iteration requiring the same amount of effort to reveal an extra digit. More precisely, one obtains an accuracy ϵ^n using only $O(n/\,\epsilon^2)$ coin tosses, or, stated differently, an accuracy $\frac{1}{A}$ with $\log A$ coins and a fixed number of tosses per coin. Of course, this classical algorithm would not work in practice, because it relies upon being given the requisite series of coins.

The quantum algorithm works in a similar way. The final complexity of the algorithm will not be limited by the number of trials, but by the fact that $O(1/\,\epsilon)$ quantum logic operations are required to "generate" the final coin (that reveals p_i to accuracy ϵ). More specifically, one generates a series of probabilities p_k and approximations E_k as follows:

$$\begin{array}{ll} & E_0 = 0 \\ p_1 \approx S & E_1 = p_1 - \frac{\epsilon_1}{2} \\ p_2 \approx S - E_1 & E_2 = E_1 + p_2 - \frac{\epsilon_2}{2} \\ p_3 \approx S - E_2 & E_3 = E_2 + p_3 - \frac{\epsilon_3}{2} \end{array} \tag{60}$$

etc. (61)

$$p_i \approx S - E_{i-1} \tag{62}$$

$$E_i = E_{i-1} + p_i - \frac{\epsilon_i}{2} \tag{63}$$

I shall now describe a quantum algorithm for estimating p_k to a given accuracy ϵ. Define $\overline{f_k} = f - E_{k-1}$. Recall that

[36] Actually, this is not precisely true. If the value of p is very close to an integral mulitple of 0.1, then one does not have confidence regarding the first digit, even though one has accuracy 0.1. Hence the estimate used in the second iteration should actually be the estimate obtained minus one half the error. However, this detail does not effect the basic principle of the algorithm.

$$p_k = S - E_{k-1} \tag{64}$$

$$= \frac{1}{M^d} \sum_{a_1,a_2,...a_d=0}^{M-1} \{f(a_1,a_2,...a_d) - E_{k-1}\} \tag{65}$$

$$= \frac{1}{M^d} \sum_{a_1,a_2,...a_d=0}^{M-1} \overline{f_k}(a_1,a_2,...a_d) \tag{66}$$

Consider a quantum computer with $d \log_2 M + 1$ qubits. Label the states $|r\rangle|a_1,a_2,...a_d\rangle$ where the first qubit r is a work qubit and the remaining qubits indicate a value in the domain of f. The computer is placed initially in the zero state: $|0\rangle|00....0\rangle$. We begin by applying a Walsh-Hadamard transform to the function qubits in order to obtain an equal superposition of all possible values for the a_i:

$$|\Psi_1 >= \frac{1}{\sqrt{M^d}} \sum_{a_1,a_2,...a_d=0}^{M-1} |0\rangle|a_1,a_2,...a_d\rangle \tag{67}$$

Next, rotate the first qubit by an amount $\overline{f_k}$. The state is then

$$|\Psi_2 >= \frac{1}{\sqrt{M^d}} \sum_{a_1,a_2,...a_d=0}^{M-1} \left(\sqrt{1 - \overline{f_k}(a_1,a_2,...a_d)^2}|0\rangle + \overline{f_k}(a_1,a_2,...a_d)|1\rangle \right) \otimes |a_1,a_2,...a_d\rangle \tag{68}$$

Finally, perform the inverse of the Walsh-Hadamard transform used in the first step. It is easy to see that the amplitude of the state $|1\rangle|00...0\rangle$ will then be p_k (because each state $|1\rangle|a_1,a_2,...a_d\rangle$ contributes amplitude $\frac{1}{\sqrt{M^d}}\overline{f_k}(a_1,a_2,...a_d)$ to the state $|1\rangle|00...0\rangle$). An estimate for p_k can therefore be obtained by making measurements of the state of the system in repeated trials, and counting the frequency of the result $|1\rangle|00...0\rangle$. To obtain an accuracy ϵ requires $O(1/\epsilon^2)$ measurements.

However, one can use amplitude amplification to increase the accuracy of the estimate. The steps described above can be viewed as a single unitary operation U that has amplitude $|U_{ts}|$ between the starting state $|s\rangle = |0\rangle|00...0\rangle$ and the target state $|t\rangle = |1\rangle|00...0\rangle$. It follows that one can use amplitude amplification to increase the probability of measuring the state $|1\rangle|00...0\rangle$. By performing only $O(N)$ operations, one can increase the amplitude of $|t\rangle$ to $N * p_k$. With the same $O(1/\epsilon^2)$ trials, one thus determines p_k with accuracy $\epsilon^* = \epsilon/N$. By fixing ϵ and choosing a large N, one performs only $O(1/\epsilon^*)$ operations to find p_k with accuracy ϵ^*.

Of course, there is a limit to the size of N if we want the amplitude of $|t\rangle$ to still be approximately $N * p_k$. (Alternatively, the size of N is limited by the requirement that $N * p_k$ remains a valid probability amplitude). This is why the algorithm requires several iterations. Initially, p_k may be any value between 0 and 1, and hence N can be at most 1. (That is, one cannot use amplitude amplification at all). As the estimates E_{k-1} become more accurate, then the value of p_k becomes correspondingly smaller, and one can choose larger and larger N.

Each estimate p_k is determined with a fixed number of trials, and since the estimates become exponentially more accurate with each E_k, the total number of trials is only a logarithmic function of the desired accuracy. Hence, the computational complexity is determined by the amplitude amplification. Within a polylogarithmic factor, the entire cost occurs on the last iteration (because each iteration takes exponentially more time). The computational complexity of the entire algorithm is therefore the same as the amplitude amplification of the last iteration: $O(1/\epsilon^*)$ operations are required, where ϵ^* is the desired accuracy.

It is interesting to note that, as with the classical Monte Carlo method, the quantum algorithm depends only upon the desired accuracy: the size of the function's domain (M^d) is irrelevant.

E. Integrals via quantum counting

There is another algorithm which can be used to evaluate the sum S in (57), inspired by the idea of quantum counting [25]. To use this method, one must first convert the real-valued function $f(a_1,a_2,...a_d)$ into a boolean valued function. This can be accomplished via the addition of an extra parameter q. The parameter takes on integral values in the range [1,Q] where Q is determined by the desired accuracy. Define

$$b(a_1,a_2,...a_d,q) = \left\{ \begin{array}{l} 1 \text{ if } q \le f(a_1,a_2,...a_d) * Q \\ 0 \text{ if } q > f(a_1,a_2,...a_d) * Q \end{array} \right\} \tag{69}$$

In other words, for a given $a_1,a_2,...a_d$,the fraction of the Q values for which $b(a_1,a_2,...a_d,q) = 1$ is the best approximation to $f(a_1,a_2,...a_d)$. It follows that the average value of b is identical to the average value of f. However, since b is a boolean-valued function, one can estimate the average value of b via approximate counting. That is, $S = \langle b \rangle = \frac{r}{M^d Q}$, where r is the number of solutions $b(a_1,a_2,...a_d,q) = 1$. To count the number of solutions r, recall that during the amplitude amplification process, the state of the system rotates within the subspace spanned by $|s\rangle$ and $U^{-1}|t\rangle$ at a rate which is proportional to $|U_{ts}|$. Moreover, recall that by using the Walsh-Hadamard transform for U (as in the Grover search algorithm), the magnitude of U_{ts} is exactly $|U_{ts}| = |W_{ts}| = 1/\sqrt{N}$ for any given target state $|i\rangle$. But if the target state $|t\rangle = \sum_{i \in b(i)=1} |i\rangle$, then the amplitude of $|U_{ts}| = r/\sqrt{N}$. Hence the amplitudes of the states $|s\rangle$ and $U^{-1}|t\rangle$ will oscillate with a frequency that varies directly with r. It is therefore a simple matter to create a superposition

$$|\Psi >= \frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |j\rangle G^j |s\rangle \tag{70}$$

and determine the value of r by performing a fast Fourier transform on the first register. The accuracy $1/A$ will depend linearly upon the number of points used in the FFT, as will the number of quantum logic operations (because it takes $O(1)$ operations to perform G, one requires $O(A)$ operations to create the state $|\Psi>$ above). It follows that one can determine the value of the integral f to accuracy ϵ with $O(1/\epsilon)$ operations, as in the previous algorithm. Also as above, one finds that the number of operations does not depend upon the size of the domain of f, but only upon the desired accuracy.

F. Discussion

At first, it may appear surprising that these two very different quantum algorithms should both require $O(1/\epsilon)$ operations. However, by exploring some variations of these algorithms, one finds that, while not identical, they are both quite similar.

First, note that there is a trivial variation of quantum counting, which is simply to measure the state of the system in repeated trials, and count the number of times one obtains the target state (or more precisely, a state for which $b(a_1, a_2, ...a_d, q) = 1$.) That is, we determine the fraction $\frac{r}{M^d Q} = \langle b \rangle = S$ through random sampling. This technique is directly analogous to the way, in Grover mean estimation, one finds the probability p_1 through repeated trials (counting the number of times we measure the target state $|1\rangle|00...0\rangle$). In both cases, $O(1/\epsilon^2)$ operations would be required to obtain an accuracy ϵ. The difference is that using the Grover method, one can subtract the most recent estimate from each term in the sum (to obtain the function $\overline{f_k}$), and then perform amplitude amplification to increase the probability of obtaining the target state. By amplifying this difference, the precision of the algorithm is limited by the linear amplitude amplification process rather than by the quadratic sampling process. In the case of quantum counting, one can also apply the amplitude amplification process to the target state (indeed, this is exactly what the quantum counting algorithm does). However, one cannot subtract the most recent estimate from each term in the sum: specifically, for a given $a_1, a_2, ...a_d$, there can be no less than zero values of q for which $b(a_1, a_2, ...a_d, q) = 1$. In the Grover method, individual terms in the sum may be negative, even though the sum of all the terms is always positive. The counting method does not allow this possibility. It is therefore impossible to use the technique of iterated, refined estimates to increase the precision of the approximation.

The relationship can be viewed from another perspective by considering a variation of Grover's method. As presented earlier, the technique depends upon measuring the amplitude of the target state $|1\rangle|00...0\rangle$. This is accomplished through repeated measurements. However, one can also determine this amplitude with a quantum FFT. Recalling once again that during the amplitude amplification process the state of the system rotates within the subspace spanned by $|s\rangle$ and $U^{-1}|t\rangle$, at a rate which is proportional to $|U_{ts}|$ (which in this case is equal to p_k), we see that one could also use an FFT to determine $|U_{ts}|$ (and therefore p_k). As in the case of quantum counting, one requires $O(1/\epsilon)$ operations to obtain the result with accuracy ϵ. Moreover, because the FFT measures the frequency of the rotation, one does not need to perform the iterated estimates (which previously ensured that the initial amplitude $|U_{ts}|$ was sufficiently small that it would in fact be amplified throughout the entire process).

The situation is in many ways similar to the relationship between Shor's algorithm and Kitaev's algorithm[61]. In the Kitaev algorithm, one estimates the phase of an eigenvalue ϕ of a unitary operator $\widehat{U}$. The number of operations required to estimate ϕ grows polynomially with the desired precision, but Kitaev obtains exponential precision by considering $\widehat{U}^2, \widehat{U}^4, \widehat{U}^8$,etc. This process is analogous to the refined estimates used in the Grover method. In [37], Cleve et. al. describe how to modify Kitaev's algorithm so that it uses an FFT to estimate the phase. The resulting algorithm is then identical to Shor's.

One sees, therefore, that the two apparently distinct algorithms are in fact both very closely related. In both cases, one performs a sequence of unitary operations that generate an operator with amplitude $|U_{ts}|$ to make a transition from the $|0\rangle$ state to the target state $|t\rangle$, where the value of $|U_{ts}|$ depends directly on the sum S. In both cases, one may use a quantum FFT to estimate the value of $|U_{ts}|$ and approximate S with accuracy ϵ in $O(1/\epsilon)$ operations. In both cases, one may estimate the value of $|U_{ts}|$ directly through repeated measurements and then approximate S with accuracy ϵ in $O(1/\epsilon^2)$ operations. The only difference is that in Grover's method, the particular form of the operator U allows one to consider negative values $\overline{f_k}$ - which in turn allows one to use the process of iterated, refined estimates and thus to obtain linear precision directly with repeated measurements instead of with the fast Fourier transform.

G. Conclusion

In conclusion: I have proposed two new applications for quantum computation: evaluating integrals and calculating descriptive statistics of stochastic processes. Whereas $O(M^d)$ operations are required on a classical deterministic Turing machine, and $O(1/\epsilon^2)$ operations are required with a classical probabilistic algorithm, one can obtain the same accuracy on a quantum computer with only $O(1/\epsilon)$ quantum operations, using two different algorithms. I have provided a simpler (and slightly more efficient) version of Grover's mean-finding algorithm, demonstrated how quantum counting can be applied to mean estimation, derived some variations of both algorithms, and shown how the two are very closely related.

It is interesting to consider these results in light of the work by Beals et. al. [8], where it is proven (using the method of polynomials) that a bounded-error quantum algorithm for computing a total function can be only polynomially more efficient than the fastest deterministic classical algorithm. A boolean function $b(a_1, a_2, ...a_d, q)$ such as the one described in Section 5 can be described as a sequence

of $M^d q$ boolean values; the average of b is a function of those $M^d q$ boolean values, and it is a total function, since it is well-defined for all possible input functions b. In order to phrase mean-estimation as a decision problem, we can ask: "Is the average value of b within the range $[E- \epsilon, E+ \epsilon]$?" (for some chosen E and ϵ). Naively, it appears that the results of [8] would imply that this problem cannot be speed up more than polynomially on a quantum computer (vs. a classical deterministic computer) - whereas I have previously claimed an exponential separation. It appears that there is a contradiction.[37]

The (in fact quite simple) resolution of this problem is that the decision question posed above does not quite correspond to mean-estimation. According to the question given, a function with mean just slightly (infinitesimally) more than $E+ \epsilon$ does not have a mean that is approximately E, whereas a function that has mean exactly $E+ \epsilon$ does. Of course, our quantum algorithms cannot reliably differentiate between these two cases in polynomial time any better than the classical deterministic algorithms can. The decision question that one can associate with mean-estimation would be a probabilistic one; the answer should be sometimes yes and sometimes no with a probability that depends (perhaps as a gaussian function) upon the distance the true mean is from the estimate E. Such a question is not a function (although it can be viewed as the average value of a weighted ensemble of functions). Thus, the results obtained in [8] do not apply to our problem, and there is no contradiction.

In concluding therefore the author would like to make the following point. It is easy for results such as those in [8] to cause one to be disheartened about the prospects of quantum computing. However, sometimes the "real" problems we wish to solve have special properties that can make them easier than the general cases. Calculating approximate integrals is one such example - and there are likely others waiting to be discovered.

V. Nonlinear Quantum Mechanics and NP-complete Problems

[38]**Summary**. This chapter will demonstrate that nonlinear quantum mechanics allows for the polynomial time solution of NP-complete and #P problems. If quantum states exhibit small nonlinearities during time evolution, then by exploiting nonlinear quantum logic gates one can design quantum algorithms that solve NP-complete and #P oracle problems. Using the Weinberg model as a simple example, the explicit construction of these gates will be derived from the underlying physics. Nonlinear quantum algorithms are also presented using Polchinski type nonlinearities which do not allow for superluminal communication.

[37]Actually, this issue applies equally to the exponential separation between the classical deterministic and probabilistic algorithms.

[38]The work described in this chapter is based upon [4] and [2].

A. *Introduction*

It has been suggested [97][98][50][65] [10] that under some circumstances the superposition principle of quantum mechanics might be violated - that is, that the time evolution of quantum systems might be (slightly) nonlinear. While there are reasons to believe that a theory of quantum gravity may involve such nonlinear time evolution, nonlinear quantum mechanics is at present hypothetical: experiments confirm the linearity of quantum mechanics to a high degree of accuracy[71][95][30][24]. (There are, however, some questions about the interpretation of these tests due to the effects of nonlinear quantum mechanics[81]). Nonlinear quantum theories have also had theoretical difficulties[79][81][54] - including problems with superluminal communication - but there are nonlinear theories that do not appear to have these issues[81].The validity of nonlinear quantum mechanics is an important question that can only be settled by further experiments and the requirements of theoretical self-consistency. However, this chapter is concerned not with the validity of a particular nonlinear theory, but instead with the implications of nonlinear quantum mechanics on the theory of computation, should quantum mechanics in fact turn out to be nonlinear at some level. In particular, I'll show that it is possible to exploit nonlinear time evolution so that the classes of problems NP and #P (including oracle problems) may be solved in polynomial time. An experimental question - that is, the exact linearity of quantum mechanics - could thereby determine the answer to what may have previously appeared to be a purely mathematical one. This chapter therefore establishes a new link between physical law and the theoretical power of computing machines, and demonstrates that the connection is much more subtle than one might suppose. Moreover, because almost all hard computational problems that occur naturally (in computer science, physics, engineering, etc.) are contained within the class of #P oracle problems, this result could (someday) be practically important as well.

As explained in Chapter 1, the class NP is (loosely defined) the set of problems for which it is possible to verify a potential solution in polynomial time. These include all problems in the class P (those that can be solved in polynomial time) as well as the NP-complete problems, e.g., traveling salesman, satisfiability, and sub-graph isomorphism, for which no known polynomial time algorithms exist. One natural way to approach these problems on a quantum computer is to create a superposition of every possible potential solution, and then try to determine if one of those potential solutions is in fact a true solution. In some sense, this technique nicely mimics the theoretical behavior of a non-deterministic Turing machine. In order to both simplify and generalize the result, it is convenient to replace the actual NP problem with an oracle problem, stated as follows: consider an oracle (or "black box") which calculates a function that maps n bits into a single bit; i.e., it takes an input between 0 and $2^n - 1$ and returns either 0 or 1. One needs to determine if there exists an input value x for which $f(x) = 1$. It is easy to see that a polynomial

time algorithm to solve this problem can be used to solve all problems in the class NP. (Note, however, that the converse is not necessarily true - the NP complete problems contain structure, whereas the function defined above is completely arbitrary. Thus this oracle problem is in fact a harder problem than those in NP, because it clearly requires exponential time on a classical Turing machine.) For simplicity we will at first restrict ourselves to the case where there is at most one value x for which $f(x) = 1$.

B. First method

One might attempt to solve this oracle problem on an ordinary quantum computer using the following technique. First, create a superposition of all the input states:

$$\psi = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, 0\rangle \tag{71}$$

Next, use the oracle to calculate $f(i)$ for each $|i\rangle$ in parallel:

$$\psi = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, f(i)\rangle \tag{72}$$

Although the final qubit in some sense "knows" the solution to the problem, a measurement of this qubit will yield $|1\rangle$ with either zero probability if there is no solution or an exponentially small probability if there is a solution. It is therefore necessary to enhance the amplitude of the $|1\rangle$ component of the superposition by an exponentially large factor, in order to distinguish the two cases. One idea is to try to increase the number of states with a $|1\rangle$ rather than increase the amplitude of the particular state $|i\rangle$ for which $f(i) = 1$. Imagine comparing the states $|i\rangle$ in pairs, according to the last bit of $|i\rangle$. Looking at just the last bit of $|i\rangle$ and the final qubit $f(i)$, we see one of the following states:

$$\begin{aligned} &(a)\ \ |00\rangle + |11\rangle \\ &(b)\ \ |01\rangle + |10\rangle \\ &(c)\ \ |00\rangle + |10\rangle \end{aligned} \tag{73}$$

The last case occurs most frequently. What we'd like to do is map these states into new states using the following transformation:

$$\begin{aligned} &(a)\ \ |00\rangle + |11\rangle \longrightarrow |01\rangle + |11\rangle \\ &(b)\ \ |01\rangle + |10\rangle \longrightarrow |01\rangle + |11\rangle \\ &(c)\ \ |00\rangle + |10\rangle \longrightarrow |00\rangle + |10\rangle \end{aligned} \tag{74}$$

This transformation is like an AND gate between branches of the wavefunction - it ignores the first qubit and places the second qubit in the state $|1\rangle$ if and only if either of the original components had the state $|1\rangle$ for the second qubit. Performing this transformation on the superposition of all $|i\rangle$ will leave every state unaffected except the state which neighbors the solution $|x\rangle$. This state will then pick up a $|1\rangle$ in place of the $|0\rangle$ which it originally had. If we then compare states in pairs according to the second bit of $|i\rangle$, the number of states with a $|1\rangle$ for the final qubit will double again. Repeated application of this process would then leave the final qubit unentangled with the first n qubits: it would be either in the pure state $|0\rangle$ if there are no solutions to the problem, or the pure state $|1\rangle$ if there had existed some state $|x\rangle$ for which $f(x) = 1$. A measurement of this qubit thereby reveals the answer to the problem.

Of course, this transformation cannot be accomplished using an ordinary quantum computer, because it is nonlinear. That this is the case can be easily seen by the fact that in cases (a) and (c) the initial states are non-orthogonal, but the final states are orthogonal. Hence the desired transformation cannot possibly be linear. One is tempted to try to patch this problem in a variety of ways. One possibility is to imbed this transformation in a larger Hilbert space and hope that a projective subspace might reduce to the desired nonlinear transformation. Unfortunately, this approach cannot succeed. The reason is that different elements of the superposition need to interfere with each other in later stages of the algorithm. If the "linearized" version of the transformation results in extraneous "garbage" qubits, these will prevent the states from interfering with each other in future iterations. Equivalently, one might hope that the non-unitary evolution associated with the measurement process might suffice to accomplish the necessary transformation, but this will fail for the same reason. One can also try to hide the extraneous information in the phases of the states. Although this appears promising at first, more careful analysis reveals essentially the same difficulties.[39]

One sees, therefore, that the potential application of a nonlinear quantum logic gate arises naturally from a fairly straightforward approach to the NP oracle problem. From an intuitive perspective, however, it is not exactly clear why it is that nonlinearity is important, beyond the fact that the gate which we desire for our algorithm does not happen to be linear. One can get a better feeling for this from a slightly different perspective.

Consider the shortest-path version of the traveling salesman problem, and a classical algorithm that finds "pretty good" solutions, such as simulated annealing. Implement this algorithm on a quantum computer, and initialize the

[39] One can also imagine approaching the problem with phases from the very beginning, thereby avoiding the need for the nonlinear gate. After calculating f(i), multiply the phase of the solution state by -1 and then reverse the computation of f(i). The computer would then be in an equal superposition of all |i>, with the state |x>for which f(x)=1 having opposite phase. Pairs of states containing two |i>of opposite phase are orthogonal to those for which both |i>are of the same phase, so it is possible to reverse the phase of the pair, thereby transfering the minus sign from the solution |x>to its partner state. Repeating the process which created the phase in the first place would then leave two states with negative phase. By iterating through each bit of |i>(as in the previous algorithm), one can continue the process until a substantial fraction of the states have negative phase. This situation can be easily detected. Unfortunately, each iteration takes twice as long as the previous one, so the algorithm described in this footnote requires exponential time.

quantum computer in a state which as before is a superposition of all possible inputs. After the algorithm has finished, the result will be a quantum computer that exists in a superposition of all the various local minima that are found by searching from every possible initial state. A measurement would then reveal any one of these local minima, but most likely not the shortest path. Thus, what one would like to do before the measurement is to compare the various states with each other and shift the amplitude into the states representing shorter paths. Put differently, we would like an algorithm which acts in a space that is restricted to only those quantum states which already have non-zero amplitude. Unfortunately, the linear transformations allowed by ordinary quantum mechanics have no way for a given state (or more precisely, component of a superposition) to "sense" the amplitude of other states. This is the aspect of nonlinear quantum mechanics which allows for the solution of NP-complete problems.

Returning to the algorithm described above, it is clear that if one could obtain the necessary nonlinear transformation, one could find the answer to an NP-complete problem in polynomial (in fact, linear) time, and using only a single evaluation of the oracle. It may be objected that the nonlinear operator described above appears arbitrary and unnatural: indeed, it was selected exactly so as to be able to solve the stated problem. However, the apparently arbitrary operation can be built using ordinary unitary operations and much simpler and more 'natural' single qubit nonlinear operators (that is, to the extent that any nonlinear operation in quantum mechanics can be considered 'natural'). One possible technique for generating the transformation would be to use the following steps: first, act on the two qubits with the unitary operator

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \\ 1 & 0 & 0 & -1 \end{bmatrix} \tag{75}$$

This transforms the states above as follows:

$$\begin{aligned} &(a)\ \frac{1}{\sqrt{2}}[|00\rangle + |11\rangle] \longrightarrow |00\rangle \\ &(b)\ \frac{1}{\sqrt{2}}[|01\rangle + |10\rangle] \longrightarrow |01\rangle \\ &(c)\ \frac{1}{\sqrt{2}}[|00\rangle + |10\rangle] \longrightarrow \frac{1}{2}[|00\rangle + |01\rangle - |10\rangle + |11\rangle] \end{aligned} \tag{76}$$

Next, operate on the second qubit with a simple one qubit nonlinear gate $\widehat{n}_-$ that maps both $|0\rangle$and $|1\rangle$to the state $|0\rangle$. Thus

$$\begin{aligned} &(a)\ \frac{1}{\sqrt{2}}[|00\rangle + |11\rangle] \longrightarrow |00\rangle \\ &(b)\ \frac{1}{\sqrt{2}}[|01\rangle + |10\rangle] \longrightarrow |00\rangle \\ &(c)\ \frac{1}{\sqrt{2}}[|00\rangle + |10\rangle] \longrightarrow |A\rangle \end{aligned} \tag{77}$$

The third final state is unknown because we have not bothered to specify how the non-linear gate acts on the state |00>+ |01>- |10>+ |11>. This omission thereby allows for flexibility in choosing the gate $\widehat{n}_-$. Whatever the state |A>may be, we can perform a unitary operation that will transform the first qubit into the pure state $|0\rangle$while leaving the state |00>in place. The computer is then in one of the following states

$$\begin{aligned} &(a)\ |0\rangle|0\rangle \\ &(b)\ |0\rangle|0\rangle \\ &(c)\ |0\rangle(x|0\rangle + y|1\rangle) \end{aligned} \tag{78}$$

A second non-linear gate $\widehat{n}_+$ is now required that will map the state $x\,|0\rangle + y\,|1\rangle$ to the state $|1\rangle$ (for the particular values of x and y which result from the above steps but not necessarily for arbitrary x and y), while leaving the state $|0\rangle$unchanged. After this gate is applied, the transformation resulting from the steps described so far is then:

$$\begin{aligned} &(a)\ \frac{1}{\sqrt{2}}[|00\rangle + |11\rangle] \longrightarrow |00\rangle \\ &(b)\ \frac{1}{\sqrt{2}}[|01\rangle + |10\rangle] \longrightarrow |00\rangle \\ &(c)\ \frac{1}{\sqrt{2}}[|00\rangle + |10\rangle] \longrightarrow |01\rangle \end{aligned} \tag{79}$$

The desired two qubit transformation is then easily obtained with a NOT gate on the second qubit and a $\pi/2$ rotation on the first qubit.

Having thus shown how to generate the needed two qubit gate, the question is now reduced to that of generating the simpler single qubit gates $\widehat{n}_-$ and $\widehat{n}_+$. If one considers the state of a qubit as a point on the unit sphere, then all unitary operations correspond to rotations of the sphere; and while such rotations can place two state vectors in any particular position on the sphere, they can never change the angle between two state vectors. A nonlinear transformation corresponds to a stretching of the sphere, which will in general modify this angle. The desired gates $\widehat{n}_-$ and $\widehat{n}_+$ are two particular examples of such operations. Excepting perhaps certain pathological cases (e.g., discontinuous transformations), it is evident that virtually any nonlinear operator, when used repeatedly in combination with ordinary unitary transformations (which can be used to place the two state vectors in an arbitrary position on the sphere), can be used to arbitrarily increase or decrease the angle between two states, as needed to generate the gates $\widehat{n}_-$ and $\widehat{n}_+$. An explicit method for generating these gates using the Weinberg model will now be provided.

C. An explicit construction using the Weinberg model

In this section, an explicit construction of the necessary nonlinear gates is provided using the Weinberg model of nonlinear quantum mechanics. Although the Weinberg model is probably not the most plausible nonlinear theory proposed to date, it serves as a good example because of its simplicity and generality, and because it is the most well-known nonlinear theory.

In Weinberg's model, the "Hamiltonian" is a real homogeneous non-bilinear function $h(\psi, \psi^*)$ of degree one, that is[98]

$$\psi_k \frac{\partial h}{\partial \psi_k} = \psi_k^* \frac{\partial h}{\partial \psi_k^*} = h \tag{80}$$

and state vectors time-evolve according to the equation

$$\frac{\partial \psi_k}{\partial t} = -i \frac{\partial h}{\partial \psi_k^*} \tag{81}$$

Following Weinberg [98], one can always perform a canonical homogeneous transformation such that a two-state system (i.e., a qubit) can be described by a Hamiltonian function

$$h = n\overline{h}(a) \tag{82}$$

where

$$n = |\psi_1|^2 + |\psi_2|^2 \tag{83}$$

$$a = \frac{|\psi_2|^2}{n} \tag{84}$$

It is easy to verify his solution to the time dependent nonlinear Schrodinger equation (81), which is

$$\psi_k(t) = c_k e^{-i\omega_k(a)t} \tag{85}$$

where

$$\omega_1(a) = \overline{h}(a) - a\overline{h}'(a) \tag{86}$$

$$\omega_2(a) = \overline{h}(a) + (1-a)\overline{h}'(a) \tag{87}$$

For nonlinear $\overline{h}(a)$, one sees that the frequencies depend on the magnitude of the initial amplitude in each basis state. Intuitively, one can imagine a transformation on the unit sphere which, instead of rotating the sphere at a particular rate, twists the sphere in such a way so that each point rotates at a rate which depends upon its angle θ from the axis (clearly, this transformation involves stretching of the surface). One can exploit this stretching of the sphere to build the gate $\widehat{n}_-$ as follows:

Step 1. Perform a rotation on the first qubit by an angle $\phi < 45°$:

$$|0\rangle \longrightarrow \cos(\phi)|0\rangle - \sin(\phi)|1\rangle \tag{88}$$

$$|1\rangle \longrightarrow \sin(\phi)|0\rangle + \cos(\phi)|1\rangle \tag{89}$$

Step 2. Time-evolve the system according to the nonlinear Hamiltonian $h = n\overline{h}(a)$. Thus

$$|0\rangle \longrightarrow \cos(\phi)|0\rangle - \sin(\phi)|1\rangle \longrightarrow \alpha\cos(\phi)|0\rangle - \beta\sin(\phi)|1\rangle \tag{90}$$

$$|1\rangle \longrightarrow \sin(\phi)|0\rangle + \cos(\phi)|1\rangle \longrightarrow \gamma\cos(\phi)|0\rangle + \delta\sin(\phi)|1\rangle \tag{91}$$

where α, β, γ and δ are phase factors. Because the initial amplitudes of the basis states are different in the two cases, the nonlinear Hamiltonian will cause the components to evolve at different frequencies. As long as these frequencies are incommensurate, there is a time t at which α=γ=δ=1 and β=-1 (to within an accuracy ε). (Further, this time t is a polynomial function of the desired accuracy ε.) The net result of these two steps is then

$$|0\rangle \longrightarrow \cos(\phi)|0\rangle + \sin(\phi)|1\rangle \tag{92}$$

$$|1\rangle \longrightarrow \sin(\phi)|0\rangle + \cos(\phi)|1\rangle \tag{93}$$

Step 3. Reverse the first step. Thus

$$|0\rangle \longrightarrow \cos(2\phi)|0\rangle + \sin(2\phi)|1\rangle \tag{94}$$

$$|1\rangle \longrightarrow |1\rangle \tag{95}$$

Essentially, we have reduced the angle between the two states by an amount 2ϕ. By suitable repetition of this procedure (that is, by choosing ϕ appropriately for each iteration), or simply by choosing ϕ precisely in the first step, the states |0>and |1>can be mapped to within ε of the state |0>, in an amount of time which is a polynomial function of the desired accuracy. This is the desired behavior for the nonlinear gate $\widehat{n}_-$. The procedure can be modified slightly to increase the angle between state vectors and produce the desired behavior for the gate $\widehat{n}_+$. With these two gates, one can solve NP-complete problems using the Weinberg model.

Note that this method is robust against small errors: the algorithm does not require exponentially precise operations at any stage.

Finally, the class #P contains problems in which you must determine not only if there exists a solution, but *the exact number* of solutions. It is easy to see that the #P problems are much harder to solve than NP-complete problems. To solve the problems in the class #P, one replaces the flag qubit with a string of $\log_2 n$ qubits and modifies the algorithm slightly - so that it adds the number of solutions in each iteration rather than performing what is effectively a one bit *AND*. In this case, a measurement of the final result reveals the exact number of solutions.

D. Second method

A different algorithm that solves the NP oracle problem can be thought of as an extension of Grover's data-base search algorithm[55] to a nonlinear regime. Suppose that it is possible to perform a nonlinear operation on a single qubit that has the following property: somewhere on the unit sphere there exists a line (of not exponentially small extent) along which application of the operation causes nearby points to move apart exponentially rapidly. One can exploit this behavior to solve NP problems in the following manner. Begin with an ordinary quantum computer (i.e., one that can perform the usual quantum logic operations) and place it in an equal superposition of all possible inputs. Then use the oracle (only once) to calculate $f(i)$ and obtain the state:

$$\psi = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i, f(i)\rangle \tag{96}$$

Now perform a $\pi/2$ rotation on each of the first n qubits. Each state $|i>$ then maps into a superposition over all possible $|i>$, with amplitude $\pm\frac{1}{\sqrt{2^n}}$. In particular, each state |i>contributes $+\frac{1}{\sqrt{2^n}}$of its amplitude to the state $|00...0>$, for a total contribution of amplitude $\frac{1}{2^n}$ from each $|i>$. At least $\frac{1}{2}2^n$ of these states correspond to a particular value of $f(i) = a$, and thus the state $|00....0, a>$ has amplitude at least $1/2$. A measurement on the first n qubits will therefore yield the state $|00...0>$ with probability at least $1/4$. The system will then be in the state

$$\psi = \frac{2^n}{\sqrt{2^{2n} - 2^{n+1}s + 2s^2}} |00..0\rangle \otimes \left\{ \frac{2^n - s}{2^n} |0\rangle + \frac{s}{2^n} |1\rangle \right\} \tag{97}$$

where s is the number of solutions i for which $f(i) = 1$. The last qubit now contains the necessary information; for small s, however, a measurement of the last qubit will almost always return $|0\rangle$, yielding no information. We wish to distinguish between the cases s=0 and s>0. This is accomplished by repeatedly applying the nonlinear operation to drive the states representing these two cases apart at an exponential rate: eventually, at a time determined by a polynomial function of the number of qubits n, the number of solutions s, and the rate of spreading, the two cases will become macroscopically distinguishable. A measurement on the last qubit will now reveal the solution. Of course, if the angular extent of the nonlinear region is small, it may be necessary to repeat the algorithm several times in order to determine the solution with high probability. In general, the algorithm will require $O((\pi/\eta)^2)$ trials, where η is the angular extent of the nonlinear region. The oracle may need to be called only once for η sufficiently large.

Problems in the class #P ask us to determine the exact number of solutions s. This is approximately found by counting the number of times that the nonlinear operator was applied. To determine s exactly, one proceeds with finer and finer estimates by rotating the final qubit such that the current best estimate is centered in the nonlinear region; in this way, applying the nonlinear operator separates states with s near this value so that they are distinguishable. With only a polynomial number of iterations, one determines the value s exactly.

Unlike the first method described previously, the above algorithm has one disadvantage in that it requires exponential precision. However, it can be made robust against noise by introducing a multiple qubit nonlinearity, as follows. Use the previous algorithm but calculate the value $f(i)$ a total of M times to obtain the state

$$\frac{2^n - s}{2^n} |000.....0\rangle + \frac{s}{2^n} |111....1\rangle \tag{98}$$

By making M sufficiently large - a constant multiple of n will suffice - the amplitude of the states with more ones than zeros (such as $|1101...1\rangle$) caused by random noise will be exponentially smaller then the amplitude caused by the existence of a single solution for which $f(i) = 1$. Hence, any nonlinear operator that rapidly increases the amplitude of states with more ones than zeroes with respect to the amplitude of states with more zeroes than ones will suffice to distinguish reliably the cases s=0 and s=1, as required. Moreover, a nonlinearity of this type satisfies the Polchinski criteria [81] for nonlinear quantum mechanics without superluminal communication. (A similar nonlinear operator is described in more detail by Czachor in [36] and was the inspiration for this approach).

E. Conclusion

In conclusion: it has been demonstrated that nonlinear time evolution can in fact be exploited to allow a quantum computer to solve NP-complete and #P problems in polynomial time. It has been shown explicitly how to accomplish this exponential speed-up using the Weinberg model of nonlinear quantum mechanics. A nonlinear quantum algorithm has also been presented using Polchinski type nonlinearities which are known not to support superluminal communication.

The author would like to emphasize that these results are probably best viewed as new and further evidence that the universe is exactly linear, rather than as blueprints for the design of a machine if it were not. (Though it is certainly not obvious, a priori, that quantum mechanics need be strictly linear - and the question can be fairly viewed as an experimental one. Moreover, it is the mere existence of a nonlinearity, no matter how small, which changes the structure of the complexity classes.) Thus, the connection between physics and computation is now made unavoidable: the underlying laws of physics strongly impact the theoretical complexity of computational problems. And while it does not seem likely that nonlinear time evolution does exist in reality, the theoretical implications and practical applications that would result from a discovery to the contrary may warrant further investigation into the matter.

REFERENCES

[1] D.S. Abrams and S. Lloyd, Phys. Rev. Lett. 79 (1997) 2586-2589

[2] D.S. Abrams and S. Lloyd, Phys. Rev. Lett. 81 (1998) 3992-3995

[3] D.S. Abrams and S. Lloyd, sub. to Phys. Rev. Lett., quant-ph/9807070

[4] D.S. Abrams and S. Lloyd, Proceedings of the 1st NASA International Conference on Quantum Computing and Quantum Communications (1998)

[5] D.S. Abrams and C. P. Williams, preprint

[6] A. Barenco et. al., Phys. Rev. A 52, 3457 (1995)

[7] A. Barenco et. al, Phys. Rev. Lett. 74, 4083 (1995)

[8] R. Beals, H. Buhrman, R. Cleve, M. Mosca, R. de Wolf, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science (1998)

[9] D. Beckman et. al, Phys. Rev. A 54, 1034 (1996)

[10] O. Bertolami, Physics Letters A, 154, p. 225-9 (1991)

[11] P. Benioff, J. Stat. Phys. 22, 563 (1980)

[12] P. Benioff, Phys. Rev. Lett. 48, 1581 (1982)

[13] P. Benioff, J. Stat. Phys. 29, 515 (1982)

[14] P. Benioff, Ann. N.Y. Acad. Sci. 480, 475 (1986)

[15] P. Benioff, Review of Quantum Computation, preprint

[16] C.H. Bennet, IBM J. Res. Dev 17, p. 523-532 (1973)

[17] C.H. Bennet, Int. J. Theor. Phys. 21, 905 (1982)

[18] C.H. Bennet, SIAM J. Comput. 18, 766-776 (1989)

[19] C.H. Bennet, IBM J. Res. Dev. 32, p. 16-23 (1988)

[20] C.H. Bennet et. al, SIAM J. Comput. 26: (5) 1510-1523 OCT 1997

[21] E. Berstein and U. Vazirani, Proceedings of the 25th Annual ACM Symposium on Theory of Computing, (ACM, New York, 1993), P. 11

[22] M. Biafore, Proceedings of the Workshop on Physics of Computation: PhysComp '94, IEEE Computer Society, Los Alamitos, CA, pp. 63-68

[23] B. Boghosian and W. Taylor, Phys. Rev. E. vol 57 (1998), p. 54

[24] J.J. Bollinger, D.J. Heinzen, W.M. Itano, S.L. Gilbert, D.J. Wineland, Phys. Rev. Lett. 63, 1031 (1989)

[25] G. Brassard, P. Hoyer, A. Tapp, quant-ph/9805082

[26] S.L. Braunstein, C.M. Caves, R. Jozsa, N. Linden, S. Popescu, and R. Schack, quant-ph/9811018

[27] G. Burkard, D. Loss, D. P. DiVincenzo, Phys. Rev. B 59, p. 2070 (1999)

[28] A. R. Calderbank and P. W. Shor, Phys. Rev. A, Vol. 54, No. 2, pp. 1098-1106, 1996

[29] A. R. Calderbank, E. M Rains, P. W. Shor, and N. J. A. Sloane, Phys. Rev. Lett. 78 (1997) 405-408

[30] T.E. Chupp and R.J. Hoare, Phys. Rev. Lett. 64, 2261 (1990)

[31] J. I. Cirac and P. Zoller, Phys. Rev. Lett. 74, 4091 (1995)

[32] S. A. Cook, Proc. 3rd Ann. ACM Symp. on Theory of Computing, Association for Computing Machinery, N.Y., 151-158 (1971)

[33] D. Coppersmith, IBM Research Report RC 19642 (1994)

[34] D.G. Cory, A.F. Fahmy, and T. Havel, Proc. Nat. Acad. Sci. USA 94, 1634-1639 (1997)

[35] D.G. Cory, private communication

[36] M. Czachor, preprint (quant-ph/9802051)

[37] R. Cleve, A. Ekert, C. Macchiavello, M. Mosca, Proc. R. Soc. Lond. A 454 (1998), pp. 339-354

[38] D.P. DiVincenzo, Science 270 255 (1995)

[39] D.P. DiVincenzo, Phys. Rev. A., 51, 1015-1021 (1995)

[40] D. Deutsch, Proc. R. Soc. London Ser. A 400, 97 (1985)

[41] D. Deutsch, Proc. R. Soc. London Ser. A 425, 73 (1989)

[42] D. Deutsch and R. Jozsa, Proc. R. Soc. Lond. A 439, 553 (1992)

[43] D. Deutsch et. al, Proc. R. Soc. Lond. Ser A 449, 669 (1995)

[44] A. Ekert and R. Jozsa, Reviews of Modern Physics 68 733 (1996)

[45] E. Farhi, J. Goldstone, S. Gutmann, M. Sipser Phys. Rev. Lett. 81 (1998) 5442-5444

[46] R.P. Feynman, Int. J. Theor. Phys. 21, 467 (1982)

[47] R.P. Feynman, Optics News 11, p. 11-20; also in Foundations of Physics 16, 507-531 (1986)

[48] R.P. Feynman, Feynman Lectures on Computation, Perseus Books (Addison-Wesley) 1996

[49] A. M. Ferrenberg and D. P. Landau, Phys. Rev. Lett. 69, 23 (1992) pp3382-3384

[50] D. Fivel, Phys. Rev. A 56, p. 146-56 (1997)

[51] E. Fredkin and T. Toffoli, Int. J. Theor Phys. 21, 219 (1982)

[52] M. Garey and D. Johnson, Computers and Intractability: A Guide to the Thoery of NP-Completeness, W.H. Freeman and Company (1979)

[53] N. A. Gershenfeld and I. Chuang, Science 275, 350-356 (1997)

[54] N. Gisin, Phys. Lett. A 113, p. 1 (1990)

[55] L.K. Grover, Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, p. 661, 212-19

[56] L.K. Grover, Phys. Rev. Lett. 79, 325-328 (1997)

[57] L.K. Grover, quant-ph/9711043

[58] W. R. Johnson, private communication

[59] B. E. Kane, Nature 393, 133 (1998)

[60] R. M. Karp, in Complexity of Computer Computations, ed. by Miller and Thatcher, PLenum Press, N.Y., p. 85-103 (1972)

[61] A. Yu. Kitaev, quant-ph/9511026

[62] R. Landauer, IBM J. Res. Dev. 5, 183 (1961)

[63] Philos. Trans. Roy. Soc. London Ser A 353, 367 (1995)

[64] D. Lidar and O. Biham, Phys. Rev. E vol.56 (1997), p.3661

[65] B.G. Levy, Physics Today, 12 pp. 20 (1989)

[66] D. Loss and D.P. Divincenzo, Phys. Rev. A.

[67] S. Lloyd, Science 261, 1569 (1993)

[68] S. Lloyd, Science 273, 1073 (1996)

[69] S. Lloyd, Phys. Rev. Lett. 75, 346 (1995)

[70] S. Lloyd, S. L. Braunstein, Phys.Rev.Lett. 82 (1999) 1784-1787

[71] P.K. Majumder et. al., Phys. Rev. Lett. 65, 2931 (1990)

[72] N. Margolus, Ann. N. Y. Acad. Sci. 480, pp. 487 (1986)

[73] N. Margolus, in Complexity, Entropy, and the Physics of Information, Santa Fe Institute Studies in the Sciences of Complexity, Vol VIII (W.H. Zurek, ed.) Addison-Wesley, pp. 273-287

[74] N. Margolus, Physica, 10D, pp. 81-95 (1984).

[75] C. Monroe et. al, Phys. Rev. Lett. 75, 4714 (1995)

[76] A. Nayak and F. Wu., quant-ph/9804066

[77] K. Obermayer, W.G. Teich, G. Mahler, Phys. Rev. B 37, 8096 (1988)

[78] G.M. Palma et. al, Proc. R. Soc. London ser. A 452, 567 (1996)

[79] A. Peres, Phys. Rev. Lett. 63, 1114 (1989)

[80] T. Pellizzari, S.A. Gardiner, J.I. Cirac, P. Zoller, Physical Review Letters, 75, pp. 3788-3791 (1995).

[81] J. Polchinski, Phys. Rev. Lett. 66, pg. 397 (1991)

[82] J. Preskill, Proc. R. Soc. Lond. A 454, 385-410 (1998)

[83] J. Preskill, unpublished notes, available at www.theory.caltech.edu/people/preskill/ph229/

[84] R. Rivest, A. Shamir, and L. Adleman (1978) Comm. ACM 21, 120-126

[85] B. Schumacher, Phys. Rev. A 51, 2738 (1995)

[86] P. Shor, in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, edited by S. Goldwasser (IEEE Computer Society, Los Alamitos, CA, 1994), p.124

[87] P. Shor, SIAM J. Computing 26 (1997) 1484

[88] D. Simon, Proceedings of the 35th Annual Symposium on Foundations of Computer Science, edited by S. Goldwasser (IEEE Computer Society, Los Alamitos, CA, 1994), p.116

[89] M. Sipser, Introduction to the Theory of Computation, PWS Publishing Company (1997)

[90] T. Sleator and H. Weinfurter, Phys. Rev. Lett. 74, 4087 (1995)

[91] A. Steane, Proc.Roy.Soc.Lond. A452 (1996) 2551

[92] Q. A. Turchette et. al, Phys. Rev. Lett. 75, 4710 (1995)

[93] A. Turing, Proc. Lond. Math. Soc. Ser. 2 42, 230-265 and 43, 544-546 (1936)

[94] W.G. Unruh, Phys. Rev. A 51, 992 (1995)

[95] R. L. Walsworth et. al., Phys. Rev. Lett. 64, 2599 (1990)

[96] W. Warren, Science 277, 1688-1690 (1997)

[97] S. Weinberg, Phys. Rev. Lett. 62, 485 (1989)

[98] S. Weinberg, Ann. of Phys. 194, pg. 336 (1989)

[99] S. Wiesner, quant-ph/9603028

[100] W. K. Wootters and W. H. Zurek, Nature 299 (1982) pp. 802

[101] A. Yao, in Proceedings of the 36th Annual Symposium on Foundations of Computer Science, edited by S. Goldwasser (IEEE computer Society, Los Alamos, CA, 1995), p. 352

[102] C. Zalka, Proc. R. Soc. Lond. A 454 (1998) pp. 313-322

[103] C. Zalka, quant-ph/9901068